(1)

Computer Graphics Research Laboratory Quarterly Progress Report No. 50

Norman I. Badler
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104-6389

Fourth Quarter 1993

January 31, 1994

140

MASTER COPY KEEP THIS COPY FOR REPRODUCTION PURPOSES

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (*Leave blank*)

2. REPORT DATE
January 1994

3. REPORT TYPE AND DATES COVERED
Technical

4. TITLE AND SUBTITLE
Computer Graphics Research Laboratory
Quarterly Progress Report No. 50

5. FUNDING NUMBERS
DAAL03-89-C-0031

6. AUTHOR(S)
Dr. Norman I. Badler

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Pennsylvania
Computer & Information Science Dept.
Philadelphia, PA 19104-6389

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
ARO 26779.22-MA-AI

11. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

This Quarterly Report includes descriptions of various projects underway in the Computer Graphics Research Lab during October through December 1993.

14. SUBJECT TERMS
Computer Graphics

15. NUMBER OF PAGES
133

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

MEMORANDUM OF TRANSMITTAL

U.S. Army Research Office
ATTN: AMXRO-RT-IPL
P.O. Box 12211
Research Triangle Park, NC 27709-2211

____	Reprint (15 copies)	XX	Technical Report (50 copies)
____	Manuscript (1 copy)	____	Final Report (50 copies)

____ Thesis (1 copy)

____ MS ____ PhD ____ Other__________

CONTRACT/GRANT NUMBER DAAL03-89-C-0031

TITLE: Computer Graphics Research Laboratory Quarterly Progress Report No. 50

is forwarded for your information.

SUBMITTED FOR PUBLICATION TO (applicable only if report is manuscript):

Sincerely,

Dr. Norman I. Badler
University of Pennsylvania
Computer & Information Science Department
Philadelphia, PA 19104-6389

DAAL03-89-C-0031

Contents

1 Introduction: Norman I. Badler

This Quarterly Report includes descriptions of various projects underway in the Computer Graphics Research Lab during October through December 1993. These reports include:

- Additional features of *Jack*®'94 (v.5.8) as well as updates on various projects underway in the Lab.
- A description of the refObject Class in *Jack*.
- An update on the integration of the collision avoidance system into *Jack* '94 (v.5.8), including torso collision avoidance.
- An introduction to path planning using the simulation of fluid flow.
- A description of a non-graphical version of *Jack* used to provide graphical information to another process or model.
- Enhancements to the Retina Window function.
- An update on the latest version of SASS, v.2.2.1, which supports the new body geometry and will be shipped as part of *Jack* '94 (v.5.8).
- An update on X-SASS (SASS under X-Windows).
- A discussion of joint center based normalization.
- A brief discussion of the female Viewpoint model To *Jack* Conversion.
- Details on the new polybody for *Jack* '94 (v.5.8).
- Progress in the area of human reach trajectory animation using human reach data supplied by MOCO corporation.
- Enhancements made to the *Jack* Animation System in the areas of textures and images.
- Plans for incorporating sensor-based navigation into *Jack*.
- A brief update on XJack.

There are also nine appendices:

- *World-Wide Active Jack Sites.*
- *Preliminary Documentation for Jack Pre-Release Features:* Welton Becket.
- *Animating Human Locomotion in Real-time Using Inverse Dynamics, Balance and Comfort Control*: Hyeongseok Ko and Norman I. Badler.
- *Blending and Morphing of Dynamic Shapes*: Douglas DeCarlo and Dimitri Metaxas.

- *ANIMATED CONVERSATION: Rule-based Generation of Facial Expression, Gesture and Spoken Intonation for Multiple Conversational Agents*: Justine Cassell, Catherine Pelachaud, Norman I. Badler, Mark Steedman, Brett Achorn, Tripp Becket, Brett Douville, Scott Prevost, Chin Seah, and Matthew Stone.
- *Texture Resampling While Ray-Tracing: Approximating the Convolution Region Using Caching*: Jeffry S. Nimeroff, Norman I. Badler, and Dimitri Metaxas.
- *Dynamic Fluid Simulations: Waves, Splashing, Vorticity, Boundaries, Buoyancy*: Nick Foster and Dimitri Metaxas.
- *Animation and Control of Four-Legged Animals*: Evangelos Kokkevis, Dimitri Metaxas, and Norm Badler.
- *Pipeline Rendering: Interactive Refractions, Reflections, and Shadows*: Paul J. Diefenbach and Norman I. Badler.

This research is partially supported by ARO DAAL03-89-C-0031 including U.S. Army Research Laboratory (Aberdeen); U.S. Air Force DEPTH through Hughes Missile Systems F33615-91-C-0001; Naval Training Systems Center N61339-93-M-0843; Sandia Labs AG-6076; DMSO through the University of Iowa; NASA KSC NAG10-0122; MOCO, Inc.; NSF IRI91-17110, CISE CDA88-22719, and Instrumentation and Laboratory Improvement Program #USE-9152503.

2 *Jack* '94 (v.5.8), Finally!: John Granieri

Jack 5.8 is finally complete! It shipped January 21st, and most everyone should receive updates soon.

I spent most of the quarter finalizing the code, and updating the User's Guide. Previous reports have detailed the highlights of this release. The User's Guide expanded by about 100 pages (to over 400 pages now!). Since there is too much new documentation to include here, please refer to the new *Jack* User's Guide for the full picture.

There are a couple of things that I put in *Jack* 5.8 at the last moment and here are short extracts from the User's Guide on these topics:

2.1 Image Formats and Textures

[1] *Jack* now has an extensible mechanism for handling different image file formats. Internally, *Jack* uses the Utah Raster Toolkit RLE format for reading and writing images (*reading*: for texture maps; *writing*: when doing things like **`write window image`**). Externally, *Jack* can read and write several different image formats, which are defined in the file **`JACK/jacklib5/jack.imagefilters.dat`**. This file defines input and output filter commands to support each image format.

Currently supported formats are:

[1] Thanks to Hanns-Oskar Porr for starting this feature.

Type	Extension	Input	Output
Utah Raster Toolkit	.rle	yes	yes
Utah Raster Toolkit (Compressed)	.rle.Z	yes	yes
SGI RGB Format	.rgb	yes	yes
SGI RGB Format (Compressed)	.rgb.Z	yes	yes
GIF	.gif	yes	yes
GIF (Compressed)	.gif.Z	yes	yes
PostScript	.ps	no	yes
PostScript (Compressed)	.ps.Z	no	yes

Figure 1: Supported Image File Formats

For each format supported, there are four lines in the filter file which define the entry:

1. Name of the format.
2. The extension of files in this format.
3. The input command. The input command must take the input filename as an argument (%s represents the filename) and produce a URT rle file on its standard output.
4. The output command. The output command will be fed a URT rle file on its standard input and produce an image file of the correct type (%s represents the filename). If either input or output is not available, this is signaled by placing the special command **not_available**.

An example entry for compressed GIF files looks like:

```
GIF (Compressed)
.gif.Z
zcat %s | giftorle
rletogif | compress | cat > %s
```

There are a set of sample textures in the **JACK/jacklib5/textures/** directory. An excellent tool for browsing the texture images is the **xv**, available from many anonymous ftp sites. [2]

2.2 Postures and Posture References

I added features to *Jack* for saving and restoring posture files which are independent of the instance of a figure (i.e. you can create posture files for one human, and apply them to any human).

[2]**xv** was also written at the University of Pennsylvania, by John Bradley at the GRASP Laboratory.

A posture block is declared within a figure block. Its purpose is to capture a posture (joint angles, figure location, root, and constraints and behaviors) for a particular figure. There are two forms of a posture declaration inside a figure block. The first one is called a *posture reference*, and uses the keyword **postureref**, and looks like:

```
figure human {
        postureref ["standing.post"] (10cm) standup;
        postureref ["sitting.post"] sitdown;
}
```

The above declaration adds two named postures to the figure *human*, and they are named *standup* and *sitdown*. The important point to know here is that a **postureref** declaration will **not** read the posture files (in this case, .**post**) when the postureref is read. It only adds the named posture (along with filename and arguments) to the list of postures attached to the figure. This is *important*, since the posture definition file (**.post** file) may contain references to peabody objects that do not yet exist in the environment. The posture file is read when a *Jack* command requests the posture to be instantiated. This also allows **postureref** declarations to be stored in figure definition files (**.fig** files).

The other form of a posture declaration is called a *posture instantiation* and looks just like above, except **postureref** is replaced with **posture**. It would look like:

```
figure human {
        posture ["standing.post"] (10cm) standup;
        posture ["sitting.post"] sitdown;
}
```

The difference here is that when the declaration is read, the posture file is read, and therefore the posture is instantiated (the figure is moved to the corresponding posture). In the above example, the figure would be moved to the *standup* posture, then the *sitdown* posture.

Posture definition files can be parameterized just like figure files and motion group files. In the example above, the posture file **standing.post** takes one parameter, a length measurement. The file **standing.post** could look like:

```
posture (length) {
        root = lower_torso.distal;
        joint left_shoulder->displacement = ...
        ...
        location = trans(length, 0cm, 0cm);
}
```

A posture file must always start with **posture**, not **postureref**. Inside a posture block, you may have joint displacements, site locations, constraints, behaviors, root specifications, etc (any field or block which may appear in a figure block may appear in a posture block). Note that all segment/site/joint names are *relative*, they are not prefixed with the figure name. This allows posture files to be *shared* between similarly structured figures (i.e. the human).

3 *Jack* and Virtual Reality: John Granieri

I am working on a posture transition system which creates a posture transition graph, and replays the transitions in real time, from recorded sequences of joint angles and figure positions. This system is used in both the Navy and Army projects to execute posture changes in real-time in their respective VR environments.

We have also built some low resolution (about 200 triangles) human figures to be used in visual simulation applications. We will make them truly "human" (in the *Jack* sense), so that they may be used as replacements for the standard human figure. This will require some re-writing of the human control code in *Jack*.

4 The refObject Class: Mike Hollick

4.1 Overview

The *refObject* class was designed to provide a facility for "linking" an arbitrary object to elements of the *Jack* environment. When an object is defined as a subclass of the refObject class, it will be notified whenever an object it is linked to is deleted. For example, the new rulers in *Jack* exhibit this behavior. Each ruler object is attached to two segments. If either segment is deleted, the ruler is notified and automatically deletes itself. The user does not have to delete the ruler before the segments, and the ruler code does not need to check the validity of both segments every cycle.

4.2 The Class

A synopsis of the refObject class is:

```
class refObject {
    short       magic;          // a magic number for validity
    int         detach_behavior;// flags the desired behavior on detach

  public:
    short       type;           // the type, for finding in a list
    List        *list;          // the list I belong to.
    List        lists;          // a list of lists I belong to.
    int         deleting_flag;  // lets detach know
```

```
                                   // if the object is being deleted
    refObject(short t);
    virtual ~refObject();
    virtual void attach(List *l);       // attach to this list
    virtual void detach(List *l=0);     // detach from l (or from list if l=0)
    virtual void attach(List *[], int); // attach to all lists in this array
                                        // of lists
    void set_detach_behavior(int i) {
        detach_behavior=i;};            // set the flag
};
```

The constructor requires an argument that identifies the type of subclass. See the example below for an illustration of this. The remaining functions are used by subclassed objects to make and break links to elements of the environment.

- **attach(List *)** ⇒ this form of the attach method allows the object to be linked to one Peabody object. The argument should be a pointer to the refObj list in the Peabody structure.
- **attach(List *[], int)** ⇒ this form allows the object to be linked with any number of Peabody objects. The first argument is an array of list pointers. Each element of the array should point to the refObj list of a Peabody object. The second argument is the number of elements in the array.
- **detach(List *l=0)** ⇒ this is used to remove the refObj's association with a Peabody object. When an argument is provided, it should be a pointer to an object's refObj list. The refObj will then be deleted from that list **only**. If no argument is given, the refObj will be deleted from the lists of all objects it is attached to. This function should be called by the subclass' destructor (see example below). This will also be called by *Jack*'s deletion routines when an object the refObj is attached to is deleted. This command will delete the refObj automatically under certain conditions, as described below:
- **set_detach_behavior(int)** ⇒ this is used to control the behavior of the **detach** function. There are two flags that can be passed:
  - **REF_NEEDS_ONE** ⇒if this behavior is specified, the refObj will be automatically deleted when it has been detached from all Peabody objects. In other words, it must be attached to at least one object.
  - **REF_NEEDS_ALL** ⇒when this is specified, the object will be deleted by the **detach** function if it is detached from **any** object. For example, the rulers use this behavior, so they are deleted automatically if either segment is deleted.

Note that the **detach** function simply deletes the refObj. Therefore, the destructor must be designed to handle two cases: one, where the refObj is deleted automatically by **detach**; and, where the refObj is deleted by something else. In the former case, the destructor is called by **detach**. In the latter case, the destructor must call **detach**. Although this may seem complicated, the flag **deleting_flag** simplifies matters and avoids any loops (see the below example).

4.3 Example 1: Rulers

Dynamic rulers are a new facility in *Jack* that allow distances to be continuously measured. The user picks a location on each of two segments (the location can be either a site or an arbitrary location). A red and white ruler is then drawn between these two points. The ruler can be deleted by the user directly, or it will be deleted when either of the two segments are deleted. The latter behavior is due to its definition as a subclass of the refObject class. The rulers were originally written before the refObject class was implemented, but changing the code to use the refObjects required only minor additions to the ruler constructor and destructor.

4.3.1 Constructor

There are actually two constructors, but for the sake of brevity I will only present one. This one takes 2 site pointers:

```
Ruler::Ruler(Site *site1, Site *site2) : refObject(REF_RULER)
{
    // assign
    seg1 = site1->segment;
    seg2 = site2->segment;

    // get the site locations (offset from segment global)
segment's    cpmatrix(off1.matrix,(GetSiteLocation(site1,NULL))->matrix);
    cpmatrix(off2.matrix,(GetSiteLocation(site2,NULL))->matrix);

    // bind and go
    sf = BindDrawer((IntFunc)draw_ruler, (void *)this);
    appendcirclist(&rulers,this);
    total_rulers++;

    // make the attachment list array
    List *list_list[2];
    list_list[0] = &seg1->refObjs;
    list_list[1] = &seg2->refObjs;

    // set the detach behavior
    set_detach_behavior(REF_NEEDS_ALL);

    attach(list_list,2);
}
```

The relevant parts of this function are the creation of the attachment list array, the detach behavior definition, and the **attach** call. Each member of the **list_list** array is simply a pointer to one of the **refObjs** circlist. The detach behavior is defined as **REF_NEEDS_ALL**. This means that the ruler object will be deleted if **either** of the segments is deleted. (If **REF_NEEDS_ONE** had been defined, the ruler would not be deleted until both segments had been deleted). Finally, the **attach** call passes the list array, along with the number of elements in the array. This attaches the object to the two segments.

Also, note the **refObject(REF_RULER)** in the constructor's declaration. This sets a flag that allows arbitrary refObject members to be identified.

4.3.2 Destructor

The destructor is straightforward:

```
Ruler::~Ruler(void)
{
    // just zap it
    UnBindDrawer(sf);
    deletecirclistdata(&rulers,this,0,0);
    total_rulers--;

    // detach from all refobj lists (if any)
    // flag needs to be set so detach() knows not to recursively delete
    // the object!
    deleting_flag = 1;
    detach();
}
```

The only thing that needed to be added was the **detach** call, along with the **deleting_flag** assignment. **deleting_flag must** be set to 1 in the destructor. Since **detach** will delete the object if the detach behavior conditions are met, the flag is necessary to notify it that the object is already in the process of being deleted.

5 Integrating the Reactive Collision Avoidance System into *Jack* '94 (v.5.8): Xinmin Zhao

During the last three months, I have been finishing the implementation of the new collision avoidance system and integrating it into *Jack* '94 (v.5.8). The new system avoids both self-collisions and collisions of the human body with environment obstacles. In this version, torso collision avoidance has been added so that collisions occurring from the pelvis and above, including the head, arms and hands, can be handled.

The new system does achieve our original goal: better performance. During interactive manipulation of a human figure, *Jack* can be running over 10 HZ (the average frame rate) with collision avoidance done automatically (measured on a Silicon Graphics workstation Crimson VGXT). Moreover, when there is no collision, the collision avoidance system causes virtually no overhead. The speed of the new system is achieved through a design that tries to minimize the overhead from the optimization routine. The new implementation also exploits spatial and temporal coherence to gain good performance. However, during the execution of motions, the performance of the system may not be as good because of attempts to stabilize the center of mass in each frame.

6 Path Planning using Laplace Equation: Xinmin Zhao

An interesting way of doing path planning is through simulating fluid flow. Imagine that we put a water source at the starting position and a sink at the goal position. The path the water follows will be a smooth path from starting to goal position. One of the simplest flows is called potential flow, which is governed by the Laplace equation (the 2D form of the equation is given below):

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = 0$$

One nice property of potential flow is that starting from any position, the flow will reach the sink if it is physically possible. In terms of path planning, this property guarantees that a collision-free path from any starting position can be found if one exists.

Potential flow can be simulated by solving the Laplace equation. One simple approach is using a finite difference method to get sets of equations and solve the equations using the successive over-relaxation (SOR) method.

Experimental results show that this method is very good for 2D path planning: multiple non-trivial paths can be planned in less than a second on a workstation. Therefore, this method can be used to do on-line path planning in a 2D workplace. The paths shown in the figure are planned in 0.3 second CPU time on a workstation.

While in theory this approach can be extended to higher dimensions, in practice, the 3D problem proves to be very expensive to solve (from over 10 minutes to hours), and the 4D problem is virtually impossible to solve. The problem is that this method requires solving very large numbers of equations (equal to the number of discrete elements). For a 4D application with 40 $\times$ 40 $\times$ 40 $\times$ 40 elements, there are 40 $\times$ 40 $\times$ 40 $\times$ 40 $= 2,560,000$ simultaneous equations to solve, which is too expensive for today's workstation, so a straightforward extension of this method to higher dimensions is not practical.

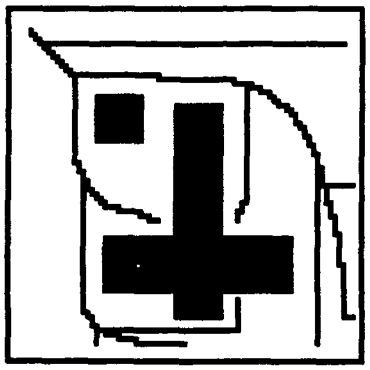


Figure 2: Path planning using the Laplace equation (the figure shows paths from various starting position to the goal position, which is located at the upper-left corner. The dark areas are obstacles.)

7 A Non-Graphical *Jack*: Zhongyang Feng

Jack is sometimes used only to provide information about the graphical environment to another process that handles the graphical display or that uses *Jack*'s human model information to control its own human model. In this case, it is not necessary for *Jack* to provide a graphical display and therefore, a version of *Jack* that has no graphical output has been developed.

While this non-graphical *Jack* is running, it reads commands from the keyboard and writes the text output and messages (informational messages and error messages) to the standard output. Although one can input command arguments interactively, commands in the syntax of standard JCL are preferred. Since it is assumed that all commands are in JCL, some functions such as argument prompt and automatic completion have been disabled because they are no longer needed.

By opening communication ports, *Jack* can also read commands through the network and run as a server that provides information and computing for external client processes. Without spending time drawing the windows, *Jack* performs operations much more quickly.

8 The Implementation of Retina Window: Zhongyang Feng

The command for opening Retina Windows was once disabled due to its poor performance. After a series of changes to its basic algorithm, it now works correctly.

The old and the current version of the Retina Window differ mainly in the following aspects:

1. The method of projecting points onto a Retina Window has been upgraded to a more accurate one. For instance, the displacement of a point in the vertical direction should also affect its horizontal coordinate in the Retina Window, and vice versa. However, the old version didn't work this way.

2. A new method of projecting the whole geometric environment onto the Retina Window has been developed in order to ensure the correct outcome. For example, in the old version, the projections of straight lines were still straight lines in the Retina Window, which in fact is wrong; and the edges which are partly out of sight are always totally ignored. These problems are all fixed in the current version by using this new method.

3. The new version opens a Retina Window for left and right eye separately, which not only shows the image more clearly, but also enables the use of the figure's own color as its color in Retina Window.

4. The figure's own color attributes, rather than the color for the left or right eye in the old version, are used as its projection's color in the Retina Window. Without this change, the Retina Window would be inaccurate when there is more than one figure in the environment.

Besides the above, there are some other minor improvements in the current version. For example, in the old version, a great deal of time is spent doing some computing whose results are ultimately rejected (e.g. computing the projections of the human figure's own body parts which won't be shown in the Retina Window). This has been improved in the new version. Moreover, no more than 10,000 nodes per psurf were permitted in the old version. There is no such limitation now.

9 SASS: Francisco Azuola

The body geometry for the polybody has been redone to improve the quality and accuracy of the model. New lower and upper limbs, head, eyes, neck, and pelvis have been developed. For now, the torso was left untouched but it will be updated for a future release.

The latest version of SASS, v.2.2.1, has been adapted to support the newest body geometry and will be shipped as part of *Jack* '94 (v.5.8). The connection between SASS and *Jack* has been tested and revised to make the interface between the programs a lot smoother.

This new version of SASS includes new features which should make using the database easier. Namely, the user can now reload information, or even load an entirely new set of data, once the program is running. Previously, data could be loaded only at start time, making it necessary to re-execute the program in order to load new data.

For the girth spreadsheet, I have included data for two more percentiles, 20 and 70, along with the already existing 1, 5, 50, 95, and 99 percentiles. In the future, data for other percentiles will be added.

The interface between SASS and the geometry, i.e., transformation factors, has been made external to the program. This means that now the user can tune the scaling to his/her own needs, without having to recompile the program. This is very useful for customizing figure production.

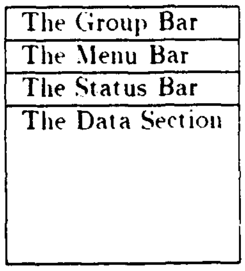
I have improved the error handler of the program to give more explicit information to the user regarding external data. I have also added some new features to the graphical interface, like a SASS hourglass for lengthy operations and more informational messages.

Finally, I've updated the user's manual with the new features for this version.

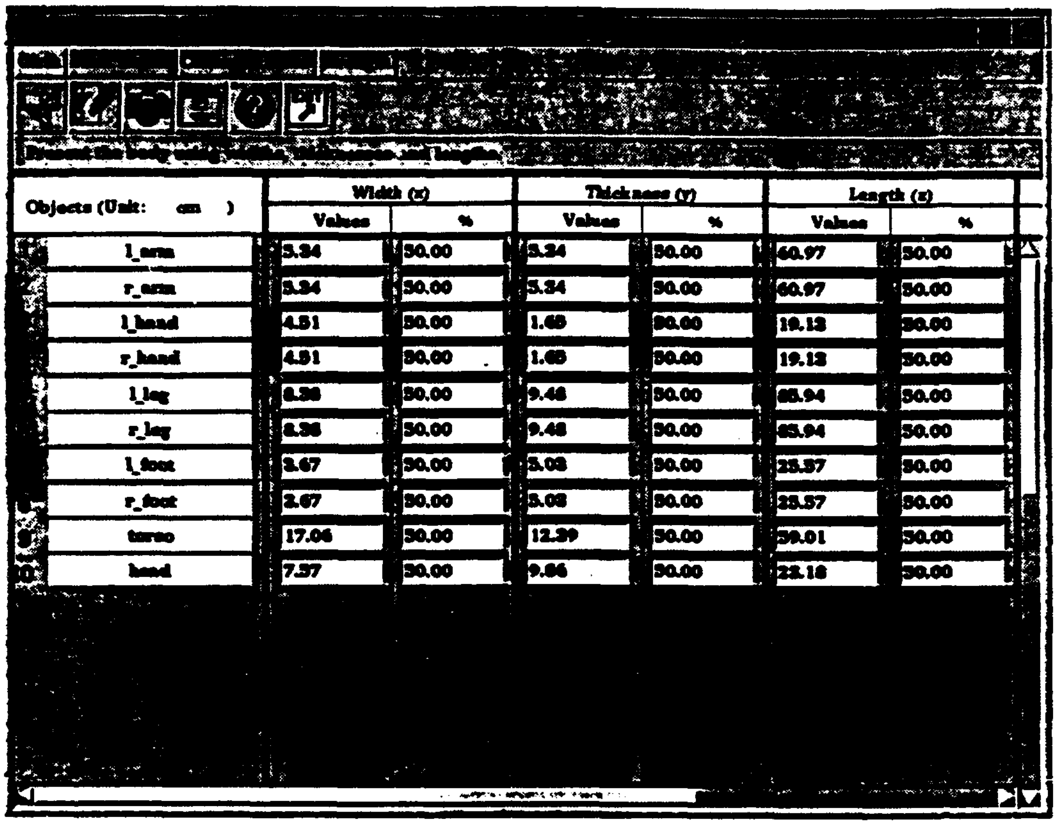
10 X-SASS: Ann Song, Francisco Azuola, Susanna Wei

X-SASS is a system that works hand in hand with SASS to create what is commonly called a graphical interface under the X environment. It behaves like a spreadsheet and also offers an interactive database query feature. Fig. 3 shows the main X-SASS screen.

The X-SASS Window has the following components:

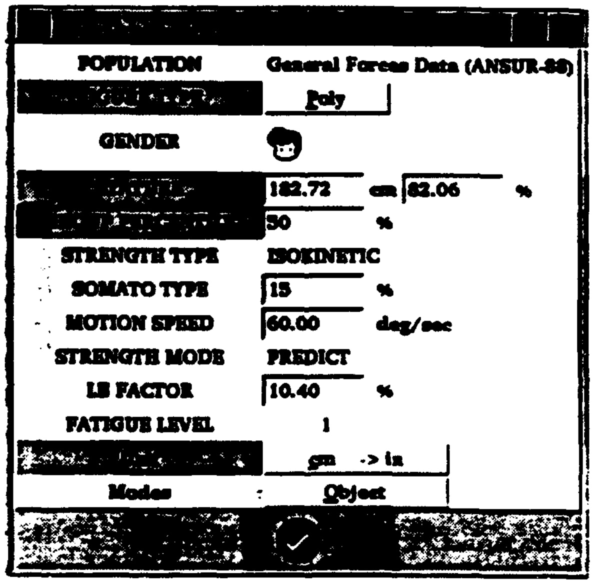


- **The Group Bar:** The Group Bar lists all of the anthropometric groups and also highlights the current working anthropometric group.
- **The Menu Bar:** The Menu Bar organizes the most common commands of the X-SASS window and it provides a way to start the applications used in X-SASS. Each menu binds to an icon button connected to a PullDown menu. The PullDown menu groups several topics together. Each topic has a single-letter mnemonic indicated by underlining. Thus, you can start an X-SASS option with the mouse or the keyboard. The six main menus are FileServer, Checklist, View, Database, Help and Exit Menus.
  - *FileServer:* FileServer is designed primarily to help the user organize files. The following functions can be performed with FileServer: Scaling Files, Figure Files, Save Sheet, Read Files.
  - *CheckList:* CheckList displays the current values of the body with entries and icons (See Fig. 4). You can control those values by simply clicking on them.
  - *View:* View has two options which are *By Label* and *By Icon*. These options determine the way the Menu Bar is displayed.
  - *DataBase:* Database is a separate application in X-SASS. The Database system contains three main components which are Menu Bar, Status Bar and Queried Data Section (See Fig. 5). The Menu Bar is on the top of the Database sheet. It has six main menus which are Query, Project, Select, Extend, Path and Back to X-SASS. The Status Bar displays the status but it also displays the current query command. The Queried Data Section is the area where the queried Data is displayed.
- **The Status Bar:** The Status Bar presents the current status and describes the current working function briefly. Most Windows applications include a Status Bar. With most menus, anytime a user is unsure of what the spreadsheet is currently working on, the user can look at the Status Bar and find the current status. The Status Bar has two main types of information, the Group Status and the Menu Status.
- **The Data Section:** The Data Section is a spreadsheet that lets the user create and edit the anthropometric group data. It can be used to produce a new human body - from thin to heavy, from short to tall. The initial data is displayed on the spreadsheet, and the entries of the spreadsheet can be modified. Each entry of the spreadsheet has its initial value and its default range. When the user types a non-numeric or an out-of-range value, a dialog box appears asking the user to fix the entry error. Every entry on the spreadsheet is one part of the human body. The user can click the leftmost column to see the information for the entry. Scrollbars can be used to scroll the spreadsheets up and down, left and right.



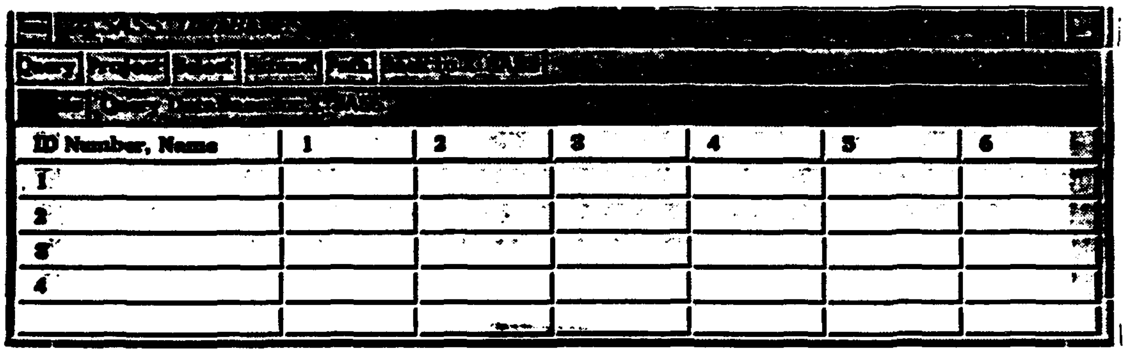
Objects (Unit: cm)	Width (x)		Thickness (y)		Length (z)	
	Values	%	Values	%	Values	%
l_arm	5.34	50.00	5.34	50.00	60.97	50.00
r_arm	5.34	50.00	5.34	50.00	60.97	50.00
l_hand	4.51	50.00	1.65	50.00	19.13	50.00
r_hand	4.51	50.00	1.65	50.00	19.13	50.00
l_leg	8.36	50.00	9.48	50.00	85.94	50.00
r_leg	8.36	50.00	9.48	50.00	85.94	50.00
l_foot	3.67	50.00	5.08	50.00	25.57	50.00
r_foot	2.67	50.00	5.08	50.00	25.57	50.00
torso	17.06	50.00	12.39	50.00	59.01	50.00
head	7.37	50.00	9.86	50.00	23.18	50.00

Figure 3: The Main X-SASS Window



POPULATION	General Forces Data (ANSUR-88)
[illegible]	Poly
GENDER	
[illegible]	182.72 cm 82.06 %
[illegible]	50 %
STRENGTH TYPE	ISOKINETIC
SOMATO TYPE	15 %
MOTION SPEED	60.00 deg/sec
STRENGTH MODE	PREDICT
LB FACTOR	10.40 %
FATIGUE LEVEL	1
[illegible]	cm -> in
Modes	Object

Figure 4: The CheckList Window



ID Number, Name	1	2	3	4	5	6
1						
2						
3						
4						

Figure 5: The Query Database Window

11 Joint Center Based Normalization: Pei-Hwa Ho

The way body segment normalization was previously done was based purely on the geometry of the segment. It assumed that each segment does not overlap with neighboring segments, that the long axis lies in the center of the segment, and that the geometry is symmetrical in certain ways. In reality, none of the human body segments satisfy these assumptions, and therefore scaling, and normalization, should be done with respect to the underlying skeleton of each segment. One way to approximate this is to use the line joining the two joint centers as the skeleton and use that as the reference in normalizing the segment.

A utility was developed so that segment normalization uses the line going through the proximal and distal sites as the long axis (z) and uses the proximal site as the origin. This will greatly preserve the appearance of models that are created from realistic geometries, e.g the Viewpoint model.

Geometries that are normalized this way need to be scaled with respect to the same axis as well and this will be built into the *Jack* "create human body" command routine. This type of scaling should only be used for the realistic models, not the standard polyhedron model.

12 Female Viewpoint To *Jack* Conversion: Pei-Hwa Ho, Leanne Hwang

We will convert the female model from Viewpoint to *Jack* format in the coming months. The only detail that we have to consider is how to handle the female upper torso so that we can have a seventeen-segment torso without endangering the integrity of the female chest.

13 New Polybody for *Jack* '94 (v.5.8): Bond-Jay Ting

This quarter, I devoted my time to creating a new polybody for *Jack* '94 (v.5.8). The hat and glasses from the old polybody were eliminated in the new polybody. This reduced the total number of segments from 71 to 69. The total number of polygons, however, has increased to 2400 due to the increased level of detail. The changes made for the new polybody are as follows:

- **Torso:** The seventeen-segment torso for the new polybody has been modified from the old polybody. The only change is that the new polybody is thicker than the old one.
- **Hands and Fingers:** Palms and fingers have also been modified from the old polybody. Finger segments remain cylindrical, while hand segments have been modified to be more human-like. Also, finger tips were added to the fingers.
- **Legs and Arms:** Similar to the old polybody, the new polybody starts with a cylindrical design, but the new polybody is tapered. The segments are no longer symmetric, but more realistic.

- **Feet and Toes:** Unlike the old polybody, the modified feet are designed without shoes on. The ridge has been moved to the inner edge of the foot. The pitch on the feet has also been made steeper.
- **Head, Neck and Eyes:** The head and neck for the new polybody are new designs. Instead of modifying the old head, the new polybody uses scanned human head and neck data as a reference. Compared to 91 polygons, the new head consists of 469 polygons. More detail was introduced such as eyebrows and ears. In the old polybody eyeballs were virtual segments, which is why the glasses were needed. In the new polybody, two 90 polygons eyeballs are added. Each eyeball has a two degree of freedom joint connected to the head segment. This improvement makes eye motion possible.

14 Human Reach Trajectory Animation: Hanns-Oskar Porr

I continued work on the MOCO corporation research and simulation project. MOCO has supplied us with a large set of experimental human reach data. This data was created by digitally sampling a human reaching motion with several Ascension Technology Bird sensors. I implemented the experimental setup in the *Jack* system, and used the constraint system in *Jack* to model the sensor data. A *Jack* segment (e.g. a forearm) is a "constraint" to the sensor data (position/orientation), meaning it tries to automatically align itself with the measurement. Since the constraint computations take a long time, I store the computed posture sets inside the new *Jack* channel structure as they become available. That way, once the computation is finished, the movement can be played back in near real-time on the screen, or put to tape. So far, I have created video tapes of animations for one test subject under two protocols (around 150 trials), and I am continuing to produce more for other subjects.

15 *Jack* Animation System: Hanns-Oskar Porr

I implemented the following new features in the *Jack* animation system:

- I extended the existing texture map facilities. In particular, I added an option that does reflection mapping, using the SGI real-time texture mapping features. It takes as input an image file that looks like a fish-eye view of an environment (many are provided), and reflects this onto any *Jack* figure.
- I added arbitrary image file support to the system. Any image file that is supported by the Utah Raster Toolkit conversion routines (e.g. gif, rgb, ps, etc.) can now be read into and written by *Jack*. This is achieved by using a UNIX pipe inside of the program. Given the file extension (e.g. ".gif"), an appropriate filter program as provided by the URT is chosen, and used in the pipe as a pre-filter before loading the file. The pipe also works for compressed files of any format.
- I added bitmap font support into *Jack*. This enables a user message to be shown on the screen in large screen font. Without this feature, any screen message always appeared blurry and

unreadable on a video tape, as the font looked too small and thin. With the new feature, the user can now type a large message on the screen that is readable on a video tape as well.

- I reorganized the texture map directory structure. First, I collected all images (in various formats) that were on the system, and then repackaged them into various sub-units (e.g. Faces, Environment maps, Textures, Places, etc). Ben Ting and I downloaded various images from image servers around the nation to expand the collection. These directories were then cataloged (hard copied), and made available to all lab members, to further aid the development of texture mapped imagery in the *Jack* system.

16 Sensor-Based Navigation: Barry D. Reich

This quarter I wrote a program which guides an agent across a field and displays the path graphically. The field is a grid of squares one meter on a side. Each grid square represents a type of terrain such as grass, water, or mud, each with a different weight. The weight is one for easily navigable terrain, such as grass, zero for impassable terrain, or somewhere in between otherwise.

The agent and target are placed at arbitrary coordinates in the grid, and the goal is for the agent to walk to the target location choosing a path with relatively little resistance. The total weight of each square adjacent to the agent is calculated by multiplying several weights together including terrain weight. The agent takes a step to an adjacent square with maximal total weight and the calculation is repeated.

With just a few weights the results are encouraging. I am now implementing this in *Jack*. A human agent will use a collection of sensors to navigate through a field, avoid obstacles, avoid enemy gaze, and arrive at a target location.

17 XJack: Ioi Lam

I have ported most of *Jack*'s functionality to XJack, a version of *Jack* that runs on X-Windows using the TK toolkit. An internal working version of XJack has been used in various Lab research projects, however, much work still needs to be done in refining the user interface and providing more intuitive interaction techniques before its release.

A World-Wide Active *Jack* Sites

- NASA - Huntsville, Marshall Space Flight Center, Alabama
- NASA-Ames Research Center, Moffett Field, California
- Computer Sciences Corporation, San Diego, California
- FMC Corporation, Santa Clara, California
- Micro Analysis and Design Inc., Boulder, Colorado
- United Technologies Research Center, East Hartford, Connecticut
- University of Delaware, Wilmington, Delaware
- NAWCTSD, Orlando, Florida
- University of Central Florida, Orlando, Florida
- McDonnell-Douglas Space Systems Company, Titusville, Florida
- US Army Infantry School, Fort Benning, Georgia
- Children Design Center, Paia, Hawaii
- Deere & Company, Moline, Illinois
- Caterpillar Inc., Peoria, Illinois
- Iowa State University, Ames, Iowa
- University of Iowa, Iowa City, Iowa
- U.S. Army Research Laboratory, Aberdeen Proving Grounds, Maryland
- National Institute of Standards and Technology, Gaithersburg, Maryland
- MIT Media Lab, Cambridge, Massachusetts
- U.S. Army Natick RD & E Center, Natick, Massachusetts

- MOCO, Inc., Scituate, Massachusetts
- General Dynamics Land Systems, Sterling Heights, Michigan
- US Army Corps of Engineers, Vicksburg, Michigan
- General Motors Corporation, Warren, Michigan
- U.S. ARMY TACOM, Warren, Michigan
- Martin Mai 'tta, Moorestown, New Jersey
- Sandia National Labs, Albuquerque, New Mexico
- Ohio State University, Columbus, Ohio
- Wright-Patterson Air Force Base, Dayton, Ohio
- University of Dayton Research Institute, Dayton, Ohio
- Anthropology Research Project, Inc., Yellow Springs, Ohio
- Traces, Inc., Bala Cynwyd, Pennsylvania
- The University of the Arts, Philadelphia, Pennsylvania
- Texas A&M University, College Station, Texas
- NASA Johnson Space Center, Houston, Texas
- Texas Woman's University, Houston, Texas
- Army Research Institute, Alexandria, Virginia
- Naval Surface Warfare Center, Dahlgren, Virginia
- NIOSH, Morgantown, West Virginia
- Kimberly-Clark Corp., Neenah, Wisconsin
- NIOSH, Finland

- Synergy Integration Ltd., Israel
- Israeli Ministry of Defense, Israel
- Japan Tech Services Corp., Japan
- Goodwin Marcus Systems Ltd., United Kingdom
- Ministry of Defence, United Kingdom
- Vickers Defence Systems, United Kingdom
- British Aerospace Dynamics, United Kingdom

B Preliminary Documentation for *Jack* Pre-Release Features: Welton Becket

B.1 Introduction

This file contains preliminary documentation for:

1. **Path walking:** realistic, curved-path walking along a user-defined path.
2. **Incremental-reactive walking:** automatic walking toward a possibly moving goal object.
3. **Dynamic strength windows:** visualization of static and dynamic torques compared with NASA 3D joint-based strength data. Computes static and dynamic torques in real-time for every joint in the body, including every joint in the spine.

These features are not being released with *Jack* '94 (v.5.8) because they are in the preliminary stages of implementation and integration. We are releasing these features early within the UPENN Computer Graphics Research Lab and, upon request, to sponsors, because they may allow human simulation and analysis not previously possible. However, please understand that the interfaces are preliminary (and thus may currently be awkward and also may change dramatically in the future), and that the commands may not be fully tested in situations and command sequences not described below.

B.2 Setup

Before you can use any of the features below, you must have the following files available somewhere in your *Jack* path:

1. `FOOTICON.fig`
2. `GOALICON.fig`
3. `GOALICON.pss`
4. `torqueicon.pss`

Make sure that one of the directories listed in your .jack5.install (in your home directory) file has the full path to all of the above files or the full path of a directory containing the above files.

In addition, you must have the full path to the following files listed in your .jack5.install file:

1. `PROTOTYPE.I`
2. `angle.knee`

B.3 Path Walking

Path provides one interface to *Jack* walking— the other is the Incremental-reactive walking described below. In path walking, the user defines a path along which the agent will walk (using the motion system). The path is defined by placing control points for a path which will be interpolated automatically (the control points are interpolated using cardinal interpolation). The resulting path will pass through all the defined control points.

To use path walking, make sure you have a human in the environment, then look at the *path walking* menu under the *contrib* menu, which contains the following commands (each described below):

- *walk along path*
- *walk from file*
- *point file to step file*
- *turn icons on*
- *turn icons off*

B.3.1 Walk along path

This command first asks you to select a human figure that is to walk. Next, it asks when the motion is to start. Note that all path walking is done in the motion system. You specify the start time of the motion and the duration is determined by the walking algorithm[3] Note that through the *Jack* lisp-API, you can determine the duration of the walk motion using the lisp command `MOTION-DUR`.

Next, you specify to goal position and orientation of the walk. A pair of feet appear that you should place and orient where you want the figure to stand at the end of the walking. The figure will step *exactly* into these feet. Note that the walking currently works only in the ground plane ($y = 0.0$)[4].

Next, you specify the curve along which to walk by placing a number of cones at control points of the curve. The first cone should be placed in the human's front hemisphere— currently, though the goal of the walk may be behind the figure's initial position/orientation, the first cone should not be placed behind the figure[5]. For each cone you wish to place, answer `1` (the default) the to the question: `more markers? no (0) or yes (1)`. You then place the cone and press **ESC** to indicate

[3]Bug 1: do not attempt to change the length of the walk motion interactively in the motion window. *Jack* will let you do this, but the walk motion will not work correctly.

[4]Bug 2: The walking should work in any plane parallel to the xz plane, though it currently does not. Also, this command should complain if the goal or the control points do not have the same y coordinate. Currently it just ignores any deviation from $y = 0.0$.

[5]Bug 3: The walking should be generalized to do a 'turn around' if the path calls for going behind the human on the first step. Currently, *Jack* will not prevent you from placing the first cone behind the human— the resulting walk in this case will be disastrous...

that you are done with the current cone. After all desired cones are placed, select `0` to the **`more markers`** prompt.

Next you are asked if you want the default walk options, and in most cases you should respond with **`yes`**. If you do select the default options, the walk command ends and the motion appears on the time-line. You can then execute the motion with the **`go`** command[6]. If you don't take the default options, the following prompts appear:

1. **`Initial Stepping Foot, left (0) / right (1):`**. Indicate whether the walk should begin with the left or right foot.
2. **`Swing Left Arm?`** `0/1`. Indicate whether the left arm should swing during the walk. If you wish to control the left arm while the figure walks, select no to this.
3. **`Swing Right Arm?`** `0/1`. Same as **`Right Arm`**.
4. **`How much (deg) to bend the torso forward`**. If you want the torso to bend forward during the walk, give an angle in degrees to bend.
5. **`How much (cm) to lower the center of mass`**. Give a distance in centimeters to lower the center of mass— this creates a 'crouch walk'.
6. **`torso control? no(0), swing(1), swing with eyecontact(2)`**. If you wish to control the upper body during the walk, you should select `0`. Selecting `1` will make the torso swing during the walk, and selecting `2` (the default) will swing the torso and make the figure look where it's about to step.

Whenever you do a walk, make sure the figure is currently in the position where the walk will begin. For example, if you want to create two walk motions in the same animation, create the first, play the animation (with **`go`**), then do the second walk from the ending position of the first walk[7].

B.3.2 Walk from file

Whenever you execute the *walk along path* command, it creates the file **`walk.steps`** in the current *Jack* directory. You can copy this file (so it doesn't get over-written), then use it as a path definition using the *walk from file* command - this allows you to describe a path and call it in as a motion later without having to specify the path again. Note that this can be done with the *walk along path* command alone by using the JCL code for the executed command[8].

This command will ask for a human figure, then ask for the start time for the motion. It then asks for a step file, which should be the name of a file generated by the *walk along path* command. It then asks if you want the default walk options— the response to this command is the same as for the *walk along path* command.

[6]Bug 4: Note that currently the goal icon (the feet) is not stored correctly by the animation system— when you play the animation the feet go to the origin. See the *turn icons off* command

[7]Bug 5: The walk motion should save the current position of the figure— currently if the figure is not in the same position at the beginning of the walk as it was when you create the walk, the walk will not work.

[8]Previously, the *walk along path* command could not be invoked using JCL— this was the main reason for providing the *walk from file* command. We have since fixed the JCL reading, however.

B.3.3 Point file to step file

This command allows you to generate a step file that can be used by *walk from file* by supplying a list of control points off-line rather than interactively. The control points and final direction at the end of the walk are supplied in a file. Each line in the file should contain a record of the form:

- **xz** *x-location z-location*

  Use this to indicate the location in the xz plane (the ground plane) of the next control point for the curve. For example the line may be:

  ```
  xz 100.0 200.0
  ```

  to place the next control point at $(100, 0, 200)$. Note that control points are ordered as they appear in the file.

- **dir** *x-location z-location*

  Use this to provide a vector that will be the direction of the human at the last control point (again, a vector in the xz plane). The direction does not have to be a unit vector. For example:

  ```
  dir 1 -1
  ```

  will make the human face along the vector $(1, 0, -1)$ at the end of the walk. The **dir** record can appear anywhere in the point file, but it must be preset. If there is more than one **dir** record, only the last is recognized.

Here is an example walk control point file:

```
xz 0 100
xz 200 300
xz 200 100
dir 1 -1
```

The command *point file to step file* first asks for a human. The current location of the human when you execute the command is used as an implied first control point— so you need not give the current location of the figure as the first control point in the point file. Then the command asks for a control point file. Then you are asked for an output file name— the name for the generated step file. This step file may then be used as input for the *walk from file* command to create a walk motion.

B.3.4 Turn icons on/off

The goal icon (the two feet) and the cones defining control points for the most recently defined walk motion are left on by default. To make them invisible, execute the command *turn icons off*. The command *turn icons on* will turn them back on. The path itself (the line on the ground plane) is currently not shut off with the *turn icons off* command— if you do not wish the path to appear during playback you must currently use the *walk from path* command, which will not create a path or any icons for the walk.

B.4 Incremental, Reactive Walking

The Incremental. Reactive Walking is another interface to the *Jack* walking. It attempts to make the figure walk to a goal object or location without requiring a path. This approach makes decisions about where the human should step at the end of the previous step. it does not compute the path in advance. Because of this. the target can be moving (interactively). and the human will keep walking until it reaches the goal. The reactive walking can be accessed through the lisp command **WALKTO**. with the following documentation (available on-line from the lisp prompt with **(help 'walkto)**:

```
WALKTO                                                      [function-doc]
Args: (human where &optional threshold)
   Human should either be a pointer to a human figure, the name of a human
   figure as a string, or the symbol $ for user input. If 'where'
   is a 3-vector the human will walk to that location, if it's a
   figure pointer or a string indicating a figure, the human will walk
   toward the base transform of the figure.  Walking stops when
   within the threshold distance (which should be in meters). This
   could be set to the target object's radius as:
        (let ((cubefig (figure-find "cube")))
        (walkto "human5" cubefig (figure-xzradius cubefig)))
   Returns an attract behavior.
```

This command walks directly toward the target object or location, so it is not possible to insure that it will walk to a particular side of the goal. To accomplish this, use two **WALKTO** commands. one to a location just infront of the goal on the desired side of the object (about 2 meters away from the object), then another to go toward the object.

This command alone will also not avoid obstacles. Obstacle avoidance can be achieved (with certain limitations) with the lisp command **AVOID**:

```
AVOID                                                       [function-doc]
Args: (human objtype &key (strength 8) (fov (/ pi 2.0)) (dist 1.5))

   Human should be a pointer to a human, the name of a human as a string,
   or $ to choose interactively.

   Objtype should be a string that is the
   type of object to avoid (any figure whose name begins with objtype will
   be avoided, ex: "cube" avoids cube, cube0, cube1, etc...).

   Strength is a relative strength factor for the avoidance.

   Fov is the field of view of the avoid sensor in radians.

   Dist is the distance to avoid and is given in meters.
```

```
Ex: (avoid "bodybuilder" "ccube" :strength 5.0 :dist 2.0)
```

An **AVOID** command should be issued for each type of object to avoid. Note, however, that you cannot avoid the object you are trying to walk to, and you cannot issue an **AVOID** for objects that are too close to the goal. Getting **AVOID** to work correctly when objects to avoid are near the goal will require testing different values for the **dist** and **strength** parameters.

AVOID is also computed incrementally, so obstacles can be moving. **AVOID** can also be used without the **WALK-TO** command— if used alone it will cause the human to walk away from approaching objects that come within the specified threshold.

NOTE: One side-effect of the incremental, reactive nature of this command is that the path taken by the human figure may be very different on different trials depending on the current configuration of objects. If you want the figure to always take the same path, use the *walk along path* command.

B.5 Dynamic Strength

The *create dynamic strength window* command creates a window that will monitor dynamic torques and available strength for a joint-chain in a human figure. In real time, the window will show:

- Current static torque for each degree of freedom
- Current dynamic torque for each degree of freedom (while in the motion system only).
- Current available torque (strength) in the positive direction.
- Current available torque (strength) in the negative direction.

Available torque is computed using **NASA Strength**[9], which is a function of the current joint angle and the current angular velocity for the joint. Once a strength window is created it will continuously display the torque and strength values during interaction and during execution of an animation[10]. The current static torque for any joint in the figure (including the spine) can be found through the *Jack* lisp-API with the lisp function **JOINT-TORQUE**. Current available strength can be found with the lisp function **JOINT-STRENGTH** (there is currently no way to get the dynamic torque— this primitive has not been implemented yet).

The commands relating to the dynamic strength are available through the *dynamic strength* menu within the *contrib* menu. It contains the following commands (described below):

1. *create dynamic strength window*
2. *dynamics go*

[9]Data courtesy of Abhilash K. Pandya of NASA Johnson Space Center
[10]Bug 6: There is currently no way to remove a strength window once created.

3. *attach load*
4. *unattach load*
5. *set load mass*

B.5.1 Create dynamic strength window

To create a strength window, first make sure a human figure is available in the environment, then execute the command *create dynamic strength window*. It will ask you to pick a human figure (if there is more than one figure). Then it will ask you for a joint group— the default is to define an arbitrary sub-chain in the figure, but if you intend to use the NASA strength data (as opposed to just creating a static/dynamic torque window), you must use only the right or left leg or arm. At the `what kind of joint group` prompt, you have the following options:

joint chain: define an arbitrary joint chain.

single joint: just one joint.

right arm: just the joints in the right arm for which there is strength data.

left arm: just the joints in the left arm for which there is strength data.

right leg: just the joints in the right leg for which there is strength data.

left leg: just the joints in the left leg for which there is strength data.

Then you are asked if you want to `show torque arrows`. Torque arrows appear on the *Jack* figure at each joint in the selected joint chain, and their lengths represent torque magnitudes. The default is not to show torque arrows.

Then you are asked for a `comparison type`, for which there are three options:

dynamics and strength: Show dynamic torque and available strength (dynamic torque becomes static torque outside of the motion system).

dynamics and statics: Show dynamic and static torque, but no strength information. This should be used for joints that do not have strength information, such as the back or the neck.

dynamics, statics, and strength: Show dynamic and static torque, and available strength in the positive and negative directions. This should only be used with a joint group type of **left arm**, **right arm**, **left leg**, or **right leg**.

The default is **dynamics, statics, and strength**.

The strength window is then created in the upper-left corner of the screen, and can be moved to wherever is convenient. The window will have a column for each degree of freedom in the joint chain. Within each column you see torques shown in $\frac{m^2 \times Kg}{sec^2}$:

- **positive available torque** (strength) as a light-purple bar above `0.0`
- **negative available torque** as dark-purple bar below `0.0`
- **current static torque** shown in green overlaid on the available strength (in the appropriate direction, positive or negative).
- **current dynamic torque** show in blue, next to the static torque.
- **exceeded available torque** is shown in red— the blue or green bars denoting static or dynamic torque turn red if they exceed available strength.

Note that when you first create a window:

- If there are no loads on the chain, torques will be very small (especially for the arms), since the torques arise only from the masses of the segments of the body. (See the *attach load* command below).
- Outside of the motion system, dynamic torque is the same as static torque (see the *dynamics go* command below).

Since the strength window operates in the background while *Jack* runs, you can manipulate the arm and see the positive and negative strength and the current static torque change[11]

On the figure, each segment involved in the selected joint chain will be highlighted to indicate percent of available torque consumed— each segment is colored from white to blue (at each joint) as current over available torque goes from 0 to 1. If current torque exceeds available torque the color turns red.

B.5.2 Dynamics go

Dynamic torques are computed only in the motion system— you must create motions on the timeline then execute the animation in order for the inverse dynamics to have a dependable sense of time for computing velocities and accelerations. Create motions for the figure as usual (dynamic torque even works during ***walk along path*** motions), but instead of using the ***go*** command to run the animation, use the ***dynamics go*** command in ***dynamic strength*** menu within the ***contrib*** menu. The ***dynamics go*** command resets the dynamics state (clears initial velocities) and makes sure attached loads are stored correctly (makes sure their animation channels are initialized correctly). Note that in the future it will not be necessary to have a different ***go*** command.

[11]Bug 7: There is currently a bug where input focus may not be set correctly after a dynamic strength window is created. If this happens (you cannot select anything in *Jack* after you've created a strength window), click somewhere outside *Jack*— get the root menu— then go back to selecting in *Jack*.

B.5.3 Attach/Unattach Load

In order to have an attached figure on the human figure considered in torque computation, use the ***attach load*** command in the ***dynamic strength*** menu. It attaches a figure to be used as a load to a site on the body (as ***attach figure*** does), and includes the figure's mass in the torque computation[12][13][14].

Use ***unattach load*** to remove a load— it takes the figure's mass out of torque computation and resets the figures root so it is no longer affixed to the attached site.

B.5.4 Set load mass

For a figure that is not attached to a human you can use the ***set segment mass*** command to change the mass of a segment. However, for a figure that has been attached as a load using ***attach load***, you must use ***set load mass*** to change the mass. Note that the default units for the entered mass is grams, but you can use other units by indicated the units explicitly (like **10kg** or **10lb**). Use this command to see sensitivity of the torques to the load.

[12]Bug 8: Currently, the ***dynamics go*** command sets channels for attached figures correctly but may disturb the channel for the human figure's root— in particular you may notice that in some situations the human's feet will slide together...

[13]Bug 9: Currently, you cannot attach more than one load to a particular site— this is an easy bug to fix, if it's a problem for anyone, let us know

[14]**Bug 9**: Currently, the mass for an attached figure is counted twice in balance computation, once for the attached figure's mass and once for the load on the site where it's attached— torques are computed correctly but the balance behavior will exaggerate the weight of the figure. Note that during walking balance is not computed so this is not a problem.

C Animating Human Locomotion in Real-time Using Inverse Dynamics, Balance and Comfort Control: Hyeongseok Ko and Norman I. Badler

Animating Human Locomotion in Real-time Using Inverse Dynamics, Balance and Comfort Control

Hyeongseok Ko, Norman I. Badler
Department of Computer and Information Science
University of Pennsylvania
200 South 33rd Street, Philadelphia, PA 19104-6389

Abstract

Human locomotion is animated with an efficient technique that performs balance and comfort control based on inverse dynamics and strength data. Loads or 3D external forces can be applied to any body point. Inverse dynamics using the Newton-Euler method is applied to a 97 DOF human model to compute the joint forces and torques in real-time. Balance is maintained by rotating or translating the pelvis and torso. The required torque at each joint is kept below the available torque based on actual human strength measurements. This comfort control adjusts the knee angle or the figure base parameters such as the step length and foot angle. The combination of the balance and comfort controls insures that dynamically sound walking motion is created in each frame. Several visualization techniques are applied to validate and display the result of the dynamics computation, such as the degree of imbalance, the ground reaction force on the foot sole, and the required vs. available joint torques. The algorithm also encompasses any walking gait, any figure scale, and any motion path.

1 Introduction

Locomotion is a major component of human activity. Many attempts using physics and dynamics have tried to reveal the principles of the process [15]. Such efforts have been continued in both computer graphics and robotics (Section 2). But many problems remain unsolved, even in characterizing the simplest case: linear, forward, rhythmic walking.

Since all mechanical linkage systems are subject to forces and the laws of physics, it is seductive to attack human motion problems with dynamics computations. Using forward dynamics, specified forces and torques allow computation of the resulting motion. This has been shown useful in predicting the motion of non-living objects [16, 42, 2, 14, 4, 29]. For example, starting from the initial state, one can simulate a swinging chain by just integrating the effect of gravity or other forces acting on the system.

Unfortunately, the theory is less successful in animating the movements of "self actuated" systems (living creatures), because the major force components – the internal muscular forces and torques – are not known *a priori* over time, and so forward dynamics cannot be used. There is no known physical law that predicts how the human body will walk (normally) nor how that walk will change if an external force acts on the model. Accordingly, it is not easy to guess the joint torque patterns that will drive the model to take a step. Even if it takes a step, the result is unlikely to resemble a human walking pattern [32, 37].

The alternative to the apparently intractable problem of specifying the joint torque patterns in advance is to use inverse dynamics in an *analysis* to compute the torques and forces that are required for the given motion. For example, the analysis can show that the motion induces excessive torque, the system is out of balance at a certain point, or the step length is too great. In this paper we present a method using the inverse dynamics computation to make the resulting walking motion dynamically balanced and the joint torques maintained within a moderate range imposed by human strength limits. This is a *correction* or *prediction* of a motion based on the inverse dynamics analysis above. Conventionally (e.g. *DADS*, *ADAMS*) the analysis and correction phases are temporally disjoint as they are designed for mechanical linkages in non-self actuated systems. First the analysis (e.g. obtaining the joint torques) may be performed for the given motion, and then the required correction (e.g. reducing the angular velocity) may be computed. Once it corrects the motion, however, the result of the analysis is not correct any more except for the very first part. The analysis should be performed again to check the dynamic soundness of the new motion. This may require many iterations to stabilize. To avoid the problem, we have built a real-time inverse dynamics package and real-time motion corrector. These are used in the system SPEEDY (Single PhasE DYnamic motion generator) described here: the analyzer and corrector are called alternately for each frame to generate dynamically sound motion in real-time.

2 Related Work

Bruderline and Calvert [8] used a kinematic parametrization of walking motion to obtain a variety of interactively specified, personalized, real-time human locomotion patterns. Boulic *et al.* [5, 6], and Ko and Badler [3, 21, 20] attempted kinematic generalization of empirical walking data to generate locomotion along a curved path and intermittent non-rhythmic stepping in any direction (forward, backward, lateral, clockwise and counter-clockwise). Their kinematic generation, however, does not handle the important case of a load or a force attached to the body.

In robotics, Vukobratović [40] simulated other parts of the body when leg motion is prescribed for level walking and stair climbing. Kajita *et al.* [18], Miura [30], and Furusho [17] built actual

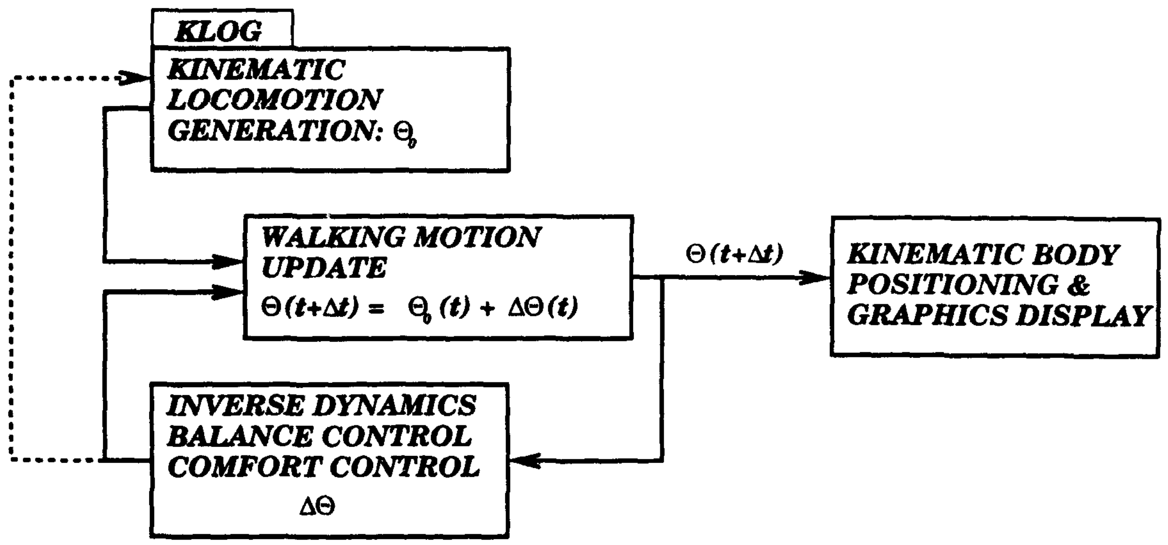

KLOG
KINEMATIC LOCOMOTION GENERATION: Θ0
WALKING MOTION UPDATE
Θ(t+Δt) = Θ0(t) + ΔΘ(t)
Θ(t+Δt)
KINEMATIC BODY POSITIONING & GRAPHICS DISPLAY
INVERSE DYNAMICS BALANCE CONTROL COMFORT CONTROL
ΔΘ

Figure 1: Overview of the Locomotion Control

biped walking robots, but walking stability was not easy to achieve. Raibert [35] built a one-legged hopping robot and controlled it to generate a stable hopping motion.

Witkin and Kass [43] used spacetime constraints to produce realistic motion of a simple articulated model of a lamp. The result conformed to traditional animation principles [23] such as anticipation, squash-and-stretch, follow through, and timing. For biped running, Girard [13] computed the impulses at each liftoff that drive the center of mass along the given path. Banking, which is a function of the velocity and curvature of running, was added for dynamic stability. McKenna and Zeltzer [28] considered the problem of dynamic control and multi-leg coordination in simulating a six-legged figure. Stewart and Cremer used a dynamics simulator *Newton*, to produce the animation of biped climbing and descending [36]. Bruderlin and Calvert used a combination of kinematic considerations and dynamic motion control for goal-directed animation of human walking [7]. Van de Panne [38] used control theory to simulate motions that entail turning, such as skiing and biking. Vasilonikolidakis and Clapworthy [39] performed inverse Lagrangian dynamics on articulated models but no animations have been noted.

3 Overview

In the SPEEDY system, human locomotion is controlled so that balance is maintained and joint stress is kept within the available torque given by empirical strength data. The latter can be used to simulate comfortable walking or walking under fatigue, by using a scaled set of available torques. Figure 1 gives an overview of the SPEEDY system, showing how a kinematic locomotion generator KLOG (Kinematic LOcomotion Generator) is used to generate walking motions which are then analyzed and dynamically modified to meet balance and stress constraints.

Inverse dynamics control requires an underlying motion Θ_0, and some mechanism to modify it over time. We use the normal gait pattern with no load for Θ_0. The modification is done through the control parameters $\Delta\Theta$ predicted through inverse dynamics. Conceptually it can be written into the equation:

$$\Theta(t + \Delta t) = \Theta_0(t) + \Delta\Theta(t). \tag{1}$$

At each frame, real-time inverse dynamics provides balance information and exerted force and

	Micro Parameters	Meta Parameters
Balance Control	$\vec{\mathbf{t}}^x_{pelvis}$ $\vec{\mathbf{t}}^y_{pelvis}$ $\vec{\mathbf{r}}^x_{torso}$ $\vec{\mathbf{r}}^y_{torso}$	lateral step width
Comfort Control	$\vec{\mathbf{t}}^z_{pelvis}$	step length foot angle
Other Purpose	$\vec{\mathbf{r}}^x_{pelvis}$ $\vec{\mathbf{r}}^y_{pelvis}$ $\vec{\mathbf{r}}^z_{pelvis}$ $\vec{\mathbf{r}}^z_{torso}$	

Table 1: The Classification of Control Parameters

torque at every joint in the model. The balance control unit computes the control parameters $\Delta\Theta(t)$ that should be added to the normal gait to retain balance. The comfort control unit compares the currently *exerted* joint torque with the *available* torque, and if it finds a strength violation, it updates some parts of $\Delta\Theta(t)$, so that the exerted torque may be reduced. If the update should be done on the movement of the figure base (foot), it is delayed until the beginning of the next step (dotted arrow in Figure 1), and affects KLOG in generating the next step. The updated pose $\Theta(t + \Delta t)$ is used to kinematically position the body for the graphics display.

4 Kinematic Locomotion Generation: KLOG

For the walking animation, biomechanical data from straight, level, forward steps is generalized to the motion of an arbitrary anthropometrically-scaled human figure. The KLOG generalization, which produces realistic locomotion animation in real-time, can take steps along any curved path, including intermittent non-rhythmic steps in any direction or turning toward any orientation. A walk can be produced simply by specifying the goal location, a path, or a walk direction. Without a path, a linear path is assumed.

Torso flexion and pelvic rotation/translation have been parameterized to generate different styles of walking. The torso can be flexed or twisted in any direction rhythmically. For the pelvis, the position as well as the orientation relative to normal walking can be controlled over the gait cycle. Combining both torso flexion and pelvic rotation/translation we can produce minor stylistic or major changes in walking.

Default values are provided if desired in setting the above parameters. Thus the user perceives KLOG as a high-level, goal-oriented locomotion system. The parameters are also modifiable through user specification or program control. Some of those parameters (*control parameters*) are used for balance and comfort control.

The control parameters are categorized into two sets: micro parameters and meta parameters (Table 1). Micro parameters $\vec{\mu}$ are related to pelvis and torso motion. They specify the relative pelvis and torso rotation and/or translation compared with the normal gait. The pelvis has six degrees of freedom: three for translation ($\vec{\mathbf{t}}_{pelvis}$) and three for the rotation ($\vec{\mathbf{r}}_{pelvis}$). The torso can bend and twist in any direction, represented by a 3-vector $\vec{\mathbf{r}}_{torso}$. Thus

$$\vec{\mu} = (\vec{\mathbf{t}}_{pelvis}, \vec{\mathbf{r}}_{pelvis}, \vec{\mathbf{r}}_{torso}). \tag{2}$$

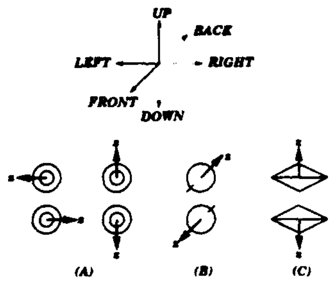


Figure 2: The Three Joint Symbols

Our conventions associate x,y, and z directions with forward, lateral (right), and down. For example, with $\vec{\mathbf{t}}_{pelvis} = (-10, 0, 10)$, the hip is both lowered and displaced backwards by 10 *cm*. If we further set $\vec{\mathbf{r}}_{torso} = (0, 30, 0)$, the torso will be bent forward by 30 degrees, and the overall motion will look like a crouched walk. Note that the balance control will adjust the gait as necessary for these torso and pelvis configurations.

Meta parameters are related to the motion of the figure base: step length, foot angle, and lateral step width relative to the normal gait. In Figure 1, the dotted arrow shows control of meta parameters, which are modified at the end of each step, to generate the next kinematic step.

As summarized in Table 1, balance control deals mostly with the micro parameters except for the lateral step width, and comfort control deals mostly with the meta parameters except $\vec{\mathbf{t}}^{z}_{pelvis}$. The micro parameters are updated every Δt, and the meta parameters are updated at the end of every walking step.

5 Real-time Inverse Dynamics

SPEEDY performs dynamic force and torque analysis at the joints of the human body model. Denavit-Hartenberg notation [12] and Newton-Euler dynamics (recursive method) [26, 11] – standard techniques in robotics – are applied. Loads or general 3D forces can be attached to the body segments. We have extra work to do to solve the closed loop problem in the lower limbs.

5.1 Dynamic Model

Owing to the efforts of J. Denavit, R. S. Hartenberg [12], and Richard P. Paul [34], a kinematic notational convention, *DH-notation*, has been established and widely accepted in robotics. Once a linked system is represented in DH-notation, its kinematics is computable in a systematic way. DH-notation is designed for a system with single DOF joints.

We model a human body as several rigid links $L_i (i = 1, \ldots, s)$ connected by joints J_j. Multiple DOF joints are decomposed into several single DOF joints in our model. For example, the hip joint is formed by cascading three revolute joints which are mutually perpendicular at the home position (standing straight up). Thus some of the L_i are null links with zero mass and length.

The three graphic symbols (adapted from Yoshikawa's book [44]) in Figure 2 are used to facilitate drawing the joints. (C) is for a *pivot* joint in which the links and rotation axis z are parallel. (A) and (B) represent the same type of *bending* joint except for a difference in view: (A) is used when the rotation axis is in left, right, up, or down in the current view; (B) is used when it is either in the front or back.

Figure 3 shows one possible assignment of coordinates in DH-convention at each joint of a human body model. The torso is modeled by seventeen vertebral discs along the spine [31]. Rotation is

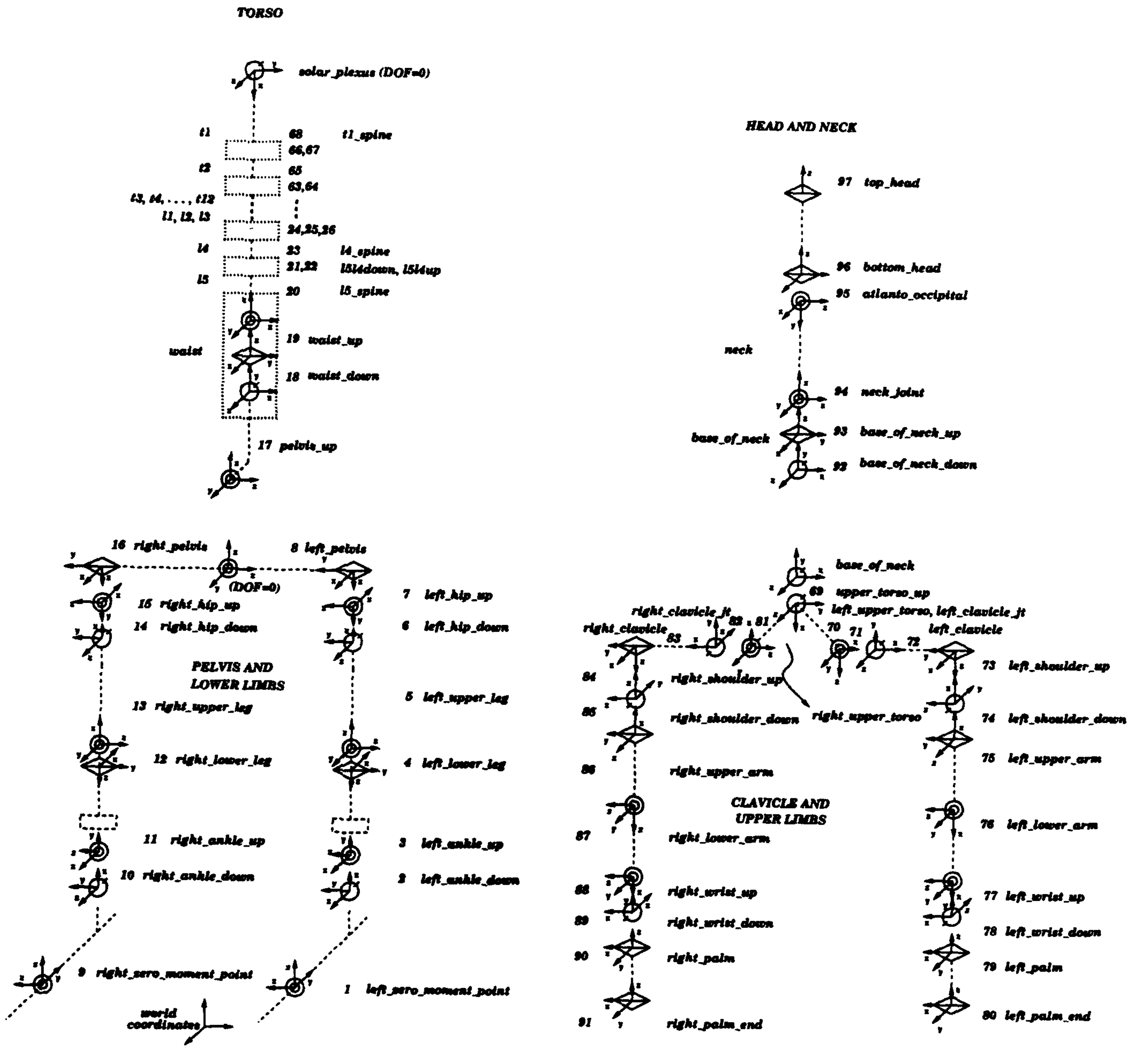


Figure 3: The Human Body Model Used in Our Dynamic Analysis

permitted only around z axes. The dotted lines in the figure represent non-null links. The dotted boxes in the lower limbs show the displaced coordinates due to DH-convention. In the torso, the dotted boxes represent the same pattern as in the three single DOF joint group at the waist.

5.2 Computation Method: Newton-Euler Method

The problem of computing the joint torque is well defined, and systematic and efficient methods exist for serial link cases. Between the two popular formulations (Newton-Euler and Lagrangian), we adapted Newton-Euler dynamics; the Newton-Euler computation costs $O(n)$ whereas Lagrangian dynamics costs $O(n^4)$ without any minimization (where n is the number of DOFs in the system). The complexity of Lagrangian dynamics can be reduced to linear time, but the coefficients are still larger than those of the Newton-Euler method. [1]

The complexity of the algorithm becomes an important factor when the model has many degrees of freedom. We have simulated several chains using *DADS* [1]. The simulation of a one or two link chain (3~6 DOFs) could be done in real-time. It took, however, 48 hours on Silicon Graphics Iris Indigo workstation to generate the animation of 20 second swing of a six link chain (18 DOFs). It would surely take more time for a human body model with 97 DOFs. Linearity of the algorithm is the minimum requirement for real-time dynamics computation of such a complex model.

The Newton-Euler method works as follows: at each time t, it computes the positional and angular acceleration of every link propagating from the base of the figure to the end-effectors (outward iteration). During the inward iteration (from the end-effectors to the base), the force and torque at the previous joint are propagated for the computation at the current joint. The mass and inertia of each link is considered during this phase of the computation. The mathematical details of the method are summarized in Appendix A.

5.3 Closed Loop Problem

We have a closed loop (in the mechanism sense) during the human figure's double stance phase: the lower limbs form a loop with the supporting plane. The difficulty in handling closed loops comes from *indeterminacy*. For example, if an object is held with both hands, inverse dynamics cannot determine the joint force and torque along an arm from the given motion alone. There can exist some counteracting forces, which are not detectable in the kinematic profile of the motion itself. This is not caused by our selection of the dynamic computation methodology; it is a generic property of the problem itself. Linear programming [27] or Moore-Penrose generalized inverse solution [19] can be applied to resolve the redundancy in a closed loop. Kumar [22], Waldron [41], and Lin [25] studied closed loop problems in cases such as multi-legged vehicles and multi-fingered grippers.

For biped locomotion, we adopted a simple approximate solution. In propagating the force and torque from the pelvis to the two thighs during the inward (Newton-Euler method) iteration, the force is distributed according to the percentage of body support on each leg. If $a\%$ of the upper body weight is supported by the left leg and $b\%$ is supported by the right leg (this can be judged by computing the center of mass and looking at its projection within in the figure's support polygon), the left hip gets $a\%$ of the force and torque from the pelvis and the right hip gets $b\%$. The details are included in Appendix B.

[1]The Lagrangian (Vasilonikolidakis and Clapworthy's method) takes $456n - 250$ multiplications and $304n - 220$ additions [39] after elaborate minimization, whereas the Newton-Euler method (our approach) in its original form takes $126n - 99$ multiplications and $106n - 92$ additions [11].

During locomotion, the dynamic computation should switch back and forth, from non-loop to loop conditions, as the loop opens or closes. Accordingly, there are three different states, LORS (Left foot Off the ground and Right leg Supporting), LSRO, and LSRS.

The closed loop problem solution may differ slightly from what is actually happening, because in practice the assumptions that were made in formulating the solution are not true during the entire duration of the motion. During locomotion, balance is relatively stable in the double stance phase. Moreover, the weight and torque from the upper body is distributed between the two legs in this phase. Thus each leg gets less stress than in a single support phase. Therefore a small discrepancy between the computation and an actual walk does not cause a drastic change in the balance and comfort control.

5.4 Approximation for Real-time Computation

The acceleration computation at t needs the states at $t - \Delta t$, t, and $t + \Delta t$. But in real-time computing, the state at $t + \Delta t$ is not available. As discussed in Section 1, to intermix the analysis with the motion correction, the computation cannot be delayed even by Δt. Thus we use instead the states at $t - 2\Delta t$, $t - \Delta t$, and t for the approximate acceleration. Even though this may create considerable differences during sharp motion changes, it appears to be quite tolerable for locomotion simulation.

5.5 Visualizations

The loading force is portrayed by a cube which swells or shrinks according to the mass (Slides A and B). Any external force is shown by a thick arrow with a blue head, which also grows or shrinks in proportion to the magnitude (Slide B). Interactive changes to either force are allowed during the motion, as demonstrated in the accompanying animation.

Three kinds of data graphs can be displayed: balance, reaction force (Slide A), and available vs. required torque (Slide B). The two windows at the bottom corners of Slide A are the balance displays: the right one is for the left leg and the left one is for the right leg. [2] The red bars show the extent of imbalance along the x, y, and z axes (subject to the conventions given in Section 3). The first bar indicates lateral imbalance, and the second indicates longitudinal imbalance. The third bar indicates the amount of twisting reaction torque from the ground to the foot sole.

The two middle windows in Slide A are the reaction force graphs. The light blue bar shows the reaction forces from the ground in the x, y, and z directions. As expected, most of the reaction force is in the z direction. Non-zero values in x or y are the reaction forces from the ground that prevent sliding, and cannot be greater than the maximum friction force.

Joints on the body display are colored (Slide B) so that the torques can be portrayed: as the torque increases, the color changes from white to blue. If the torque exceeds the strength limit, the color is set to red. The thin arrow (with red head) coming out of the joint indicates the magnitude of the torque.

The two bar graphs in Slide B are the available vs. required torque panels of the right and left legs. A bar corresponds to a DOF. There are seven DOFs in a panel: three for the hip, one for the knee, three for the ankle. The purple bar shows the available torque (strength). Each DOF can rotate in two directions, positive and negative. The strengths in these two directions are different, and are called the positive and negative strengths, respectively. The light purple bar shows the positive strength, and the dark purple bar shows the negative strength. The blue bar shows the

[2]The apparent left-right reversal is because we like to view the figure walking towards us; thus its left is our right. The window thus appears in a visually compatible position on the screen.

dynamic required torque for the motion. The green bar shows the static torque considering each frame as a static case. The blue (dynamic torque) and green (static torque) bars can grow or shrink within the purple bar (strength). If the blue or green bar exceeds the purple bar, that part is colored red, indicating a strength violation. At the same time, the joint of the human figure on the display turns red. The available vs. required torque panels can be created for other parts of the body.

6 Balance Control

6.1 The Zero Moment Point and Balance

Static balance can be achieved by keeping the projection of the center of mass within the figure's support polygon, even if only one leg provides support. In locomotion, however, we need to consider dynamic balance because the inertia effect is not negligible.

Consider a single support phase. Even though the body seems to be supported by the whole foot, we can find a point on the sole at which the moment is zero. This idea comes from the analogy of walking "on tiptoes". There cannot be any exerted torque at the toe. Similarly during walking there is a point called ZMP (zero moment point) [40] where the exerted torque should be zero. ZMP is not a fixed point but it translates from the heel to the tip of the toe during support. We create a fake joint at ZMP that connects the foot to the world.

6.2 Balance Vector and Balance Control

Normally the moment at ZMP should stay zero all the time. If the result of inverse dynamics indicates a non-zero value, it means such a torque should have been exerted for the motion. We interpret it as the measure of imbalance at that moment. We call the torque at the ZMP the *balance vector* $\vec{b}$. Thus, for example, if the balance vector is heading forward (Figure 4), a left-to-right torque (by the right hand rule) should have been exerted to prevent lateral right-to-left collapse. Therefore, moving the pelvis to the perpendicular direction (right side) of $\vec{b}$, and bending the torso left-to-right will help to keep the balance. If we let $\vec{b}^{\perp}$ be the vector obtained by rotating $\vec{b}$ by 90 degrees clockwise, the pelvis displacement is given by

$$\Delta PELVIS = \alpha\vec{b}^{\perp}, \tag{3}$$

and the torso bending is given by

$$\Delta TORSO = \beta\vec{b}. \tag{4}$$

Determining the value of α and β is not an easy problem. If we use values that are too large, the adjustment will overshoot the correct balance, and if too small, the adjustment will be too slow to achieve balance in time. Also, the ratio between them must be determined. These issues will be addressed in Section 7.

During the double stance phase, the balance vector is approximated by the weighted average of the balance vectors at the two feet according to the relative position of the center of mass projection.

6.3 Lateral Swaying

Balancing creates lateral swaying. The swaying amplitude increases as the load gets heavier, the step gets slower, or the lateral width of the step gets wider. Swaying causes energy consumption (as kinetic energy). It can be reduced by having laterally narrower steps. Thus after each step, if the step caused too much swaying, the lateral step width is reduced by a certain amount. As the

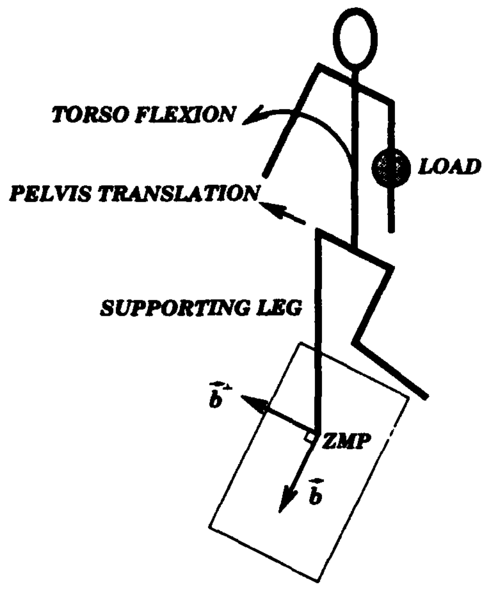


Figure 4: The Balance Vector

load gets heavier, we use a narrower lateral step width. We can even observe a negative lateral step width in real human walks when an excessive load is carried.

6.4 Compensation along the Spine

We use 17 segment torso model. There are 17 flat discs connected along the spine. This skeletal model creates a torque at the waist to the forward direction. A more complex abdominal strength model [10, 9] may solve the problem, but it would require more complex computations. Instead, we just added a counteracting torque backwards to compensate.

7 Comfort Control

7.1 Strength Data

Obviously there is a limit to the torque that can be exerted at a joint. For each rotation axis, we have two limits: one for extension and the other for flexion. That is, the body torque that tries to win the external flexional torque is called ***extensional torque*** and have a negative value. The ***flexional torque*** is similarly defined and has a positive value. Thus the torque at each joint is limited to the extreme values of the extensional and flexional torques. These upper and the lower bounds are determined from strength data.

Pandya et. al. [33]. collected strength data of several human subjects. Strength is characteristic of each individual; moreover, it depends on joint angles and their angular velocity. It is approximated by a second degree polynomial:

$$y = f_0(\dot{\theta}) + f_1(\dot{\theta})\theta + f_2(\dot{\theta})\theta^2 \tag{5}$$

where θ is the joint angle, and f_0, f_1, and f_2 are the functions of $\dot{\theta}$.

Lee *et al.* [24] proposed an animation technique that considers comfort at the joint during an end-effector motion. The end-effector proceeds for Δt along a suggested direction (e.g. straight

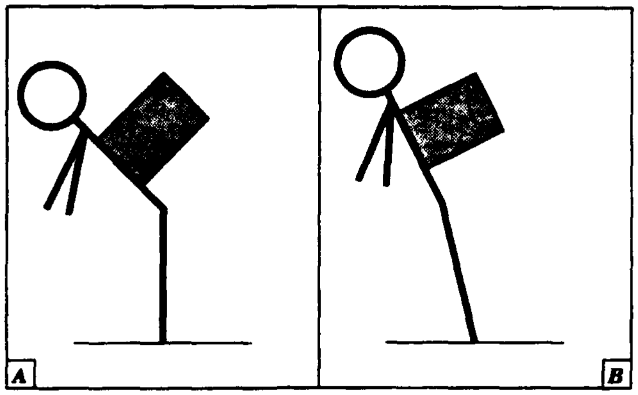


Figure 5: The Knapsack Example

path to the goal) unless it causes a strength violation. If that occurs, a cone of possible directions is computed and the original direction is projected within the cone. For example, if the object in a lifting task is heavy, the hand trajectory is altered to move it closer to the body to reduce the required torque. Since locomotion, however, moves the figure base as well as the end-effectors, a more complex strategy is required.

7.2 Comfort Control

If the system is balanced on a point, the overall moment around that point is zero by the definition of dynamic balance. For example, if a knapsack is attached at the back, the upper body alone can achieve a balance by bending the torso forward an appropriate amount (Figure 5 A). In this case the torque at the waist is zero, and the torques at the joints along the lower limbs will be smaller than in the case where the upper body balance is not at the waist (Figure 5 B). This observation leads us to balance by torso flexion only and suggests a relatively large value for β. But too much torso bending can lead to an excessive instantaneous torque at the waist at the moment of heel strike.

The imbalance at the waist can be corrected by translating the pelvis (Figure 5 B). In this case the exerted torque at the waist will not be zero. Moreover, excessive pelvis translation can lead to a problem in locomotion. For instance, if the pelvis is translated excessively to the right, the left leg may be too short for support during the right step. The problem is fixed by having a laterally narrower step, and/or translating the pelvis back to the left with more torso flexion.

We used $\alpha = 0.02$ and $\beta = 0.018$ for the accompanying real-time animation. Note that the values are not compatible by themselves: one is positional (cm) and the other is angular (degrees). The pelvis translation was limited to $5cm$, $10cm$, and $2cm$ in x, y, and z directions, respectively. Thus if there is excessive external force in the lateral direction, the pelvis is displaced by $10cm$, and the rest of the balance is achieved by torso bending.

Comfort control manages the step length, foot angle, and knee angle parameters. A shorter step together with a smaller foot angle variation induces less torque at the hip, thus less torque at the other joints along the lower limbs. A smaller knee angle will help to reduce the torque at the knee.

After each step, if any strength violation is detected, the step length and foot angle are reduced by a certain amount. The knee angle is indirectly controlled through the pelvis height $\vec{t}^z_{pelvis}$. Thus the pelvis is raised a certain amount during the stance phase, creating a more or less stiff walking pattern. In the accompanying animation, we can observe the decrease of the step length, foot angle, and knee angle until the problem (red colored joint) disappears.

8 Discussion

An excessive pelvis translation requires a pelvis rotation, to compensate for the fixed length of the leg. For example, if the pelvis is displaced laterally, the pelvis needs a rotation around the x direction. Thus the *other purpose* parameters in Table 1 are used to set the proper pelvis orientation together with the balance control parameters.

In the dynamic simulation, a variable load and/or a 3D force can be attached at any point of the body. If a load is attached, the mass and inertia of the attached link is recomputed. As shown in the appendix, we assume the external force is acting on the center of mass. If not, we translate it to the COM by adding an extra moment on the link.

Our decision in the comfort control is based on the comparison between the required and the available torques. The joint forces should be also considered for a dynamically safe motion. The joint force is defined to be the force exerted at the joint from the above link. In a situation that involves a great impact, the joint force becomes more important. When landing from a high jump, a stiff straight legs induce zero torques but huge forces at the moment of the impact. The joint impact force, however, is partly reflected in the joint torques during locomotion, as summarized in Appendix A: there is a cross product term which happens to be zero in a stiff leg landing. Normally impact forces are not great enough to cause problems in locomotion. Our decisions on strength violation may be refined later if we acquire data on impact endurance.

SPEEDY is implemented on a Silicon Graphics Iris Crimson workstation. It draws 20 frames for the curved path walking alone. With the balance and comfort control, it draws 10 frames per second. The speed is reduced by half if about 2000 polygon faces should be shaded.

The SPEEDY system for human locomotion animation implements an efficient real-time technique using balance and comfort control, inverse dynamics, and strength data. It modifies the walk in real-time when a load or a 3D external force is applied. Several visualization techniques display the result of the dynamics computation.

The accompanying video has three parts. The first part shows the gait changes due to the balance control. The second part shows the result of the comfort control. The final part is a short movie in which a person carries a dog in a strong wind (Slide C). The inflexible chain was added to show the direction and magnitude of the wind: the chain direction shows the direction away from the wind, and the (world coordinate) chain angle shows the magnitude of the wind. The dog's motion does not affect the chain. The piece demonstrates realistic human locomotion and postural adjustment in the presence of significant loads and changing external force.

References

[1] Computer Aided Design Software, Inc. DADS tutorial, dynamic analysis and design system, revision [illegible].0.

[2] W. W. Armstrong and M. Green. The dynamics of articulated rigid bodies for purposes of animation. In M. Wein and E. M. Kidd, editors, *Graphics Interface '85 Proceedings*, pages 407–415. Canadian Inf. Process. Soc., 1985.

[3] Norman I. Badler, Cary B. Phillips, and Bonnie L. Webber. *Simulating Humans: Computer Graphics Animation and Control.* Oxford University Press, 1993.

[4] Ronen Barzel and Alan H. Barr. A modeling system based on dynamic constraints. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 179–188, August 1988.

[5] Ronan Boulic, Nadia Magnenat-Thalmann, and Daniel Thalmann. A global human walking model with real-time kinematic personification. *The Visual Computer*, 6:344–358, 1990.

[6] Ronan Boulic and Daniel Thalmann. Combined direct and inverse kinematic control for articulated figure motion editing. *Computer Graphics Forum*, 11(4):189–202, 1992.

[7] Armin Bruderlin and Thomas W. Calvert. Goal-directed, dynamic animation of human walking. *Computer Graphics*, 23(3):233–242, July 1989.

[8] Armin Bruderlin and Tom Calvert. Interactive animation of personalized human locomotion. In *Proceedings of Graphics Interface '93*, pages 17–23, Toronto, Canada, May 1993.

[9] D.B. Chaffin. Software development for high exertion manual jobs. In *1993 International Industrial Engineering Conference Proceedings*, pages 285–294, Los Angeles, May 1993.

[10] D.B. Chaffin and M. Erig. Three-dimensional biomechanical static strength prediction model sensitivity to postural and anthropometric inaccuracies. *IIE Transactions*, 23(3):215–227, 1991.

[11] John J. Craig. *Introduction to Robotics, Mechanics and Control.* Addison-Wesley, second edition edition, 1989.

[12] J. Denavit and R. S. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *ASME Journal of Applied Mechanics*, 77:215–221, 1955.

[13] Michael Girard. Interactive design of 3D computer-animated legged animal motion. *IEEE Computer Graphics and Applications*, 7(6):39–51, June 1987.

[14] James K. Hahn. Realistic animation of rigid bodies. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 299–308, August 1988.

[15] Verne T. Inman, Henry J. Ralston, and Frank Todd. *Human Walking.* Williams and Wilkins, Baltimore/London, 1981.

[16] Paul M. Issacs and Michael F. Cohen. Controlling dynamic simulation with kinematic constraints, behavior functions and inverse dynamics. In *Proceedings of SIGGRAPH*, 1987.

[17] Furusho J. and M. Masubuchi. Control of a dynamic biped locomotion system for steady walking. *Journal of Dynamic Systems, Measurement, and Control*, 108:111–118, June 1986.

[18] Shuuji Kajita, Kazuo Tani, and Akira Kobayashi. Dynamic walk control of a biped robot along the potential energy conserving orbit. *IEEE International Workshop on Intelligent Robotics and Systems, IROS '90*, pages 789–794, 1990.

[19] C. A. Klein, K. W. Olson, and D. R. Pugh. Use of force and attitude sensors for locomotion of a legged vehicle over irregular terrain. *The International Journal of Robotics Research*, 2(2):3–17, 1983.

[20] Hyeongseok Ko and Norman I. Badler. Intermittent non-rhythmic human stepping and locomotion. In *Proceedings of the First Pacific Conference on Computer Graphics and Applications, Pacific Graphics '93*, pages 283–300, Seoul, August 1993. World Scientific.

[21] Hyeongseok Ko and Norman I. Badler. Straight line walking animation based on kinematic generalization that preserves the original characteristics. In *Graphics Interface '93*, Toronto, Canada, May 1993.

[22] Vijay R. Kumar and Kenneth J. Waldron. Force distribution in closed kinematic chains. *IEEE Journal of Robotics and Automation*, 4(6):657–664, 1988.

[23] John Lasseter. Principles of traditional animation applied to 3D computer animation. *Computer Graphics*, 21(4):35–44, July 1987.

[24] Philip Lee, Susanna Wei, Jianmin Zhao, and Norman I. Badler. Strength guided motion. *Computer Graphics*, 24(4):253–262, August 1990.

[25] Yueh-Jaw Lin and Shin-Min Song. A comparative study of inverse dynamics of manipulators with closed-chain geometry. *Journal of Robotic Systems*, 7(4):507–534, 1990.

[26] J. Y. S. Luh, M. W. Walker, and R. P. Paul. On-line computational scheme for mechanical manipulators. *Transactions of the ASME Journal of Dynamic Systems, Measurement, and Control*, 1980.

[27] R. B. McGhee and D. E. Orin. A mathematical programming approach to control of joint positions and torques in legged locomotion systems. In *Proceedings of ROMANSY-76 Symposium*, Warsaw, Poland, September 1976.

[28] Michael McKenna and David Zeltzer. Dynamic simulation of autonomous legged locomotion. *Computer Graphics*, 24(4):29–38, August 1990.

[29] Dimitri Metaxas and Demetri Terzopoulos. Dynamic deformation of solid primitives with constraints. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 309–312, July 1992.

[30] Hirojumi Miura and Isao Shimoyama. Dynamic walk of a biped. *The International Journal of Robotics Research*, 3(2):60–74, Summer 1984.

[31] Gary Monheit and Norman I. Badler. A kinematic model of the human spine and torso. *IEEE Computer Graphics and Applications*, 11(2), 1991.

[32] J. Thomas Ngo and Joe Marks. Spacetime constraints revisited. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 343–350, August 1993.

[33] Abhilash K. Pandya, James C. Maida, Ann M. Aldridge, Scott M. Hasson, and Barbara J. Woolford. The validation of a human force model to predict dynamic forces resulting from multi-joint motions. Technical Report 3206, NASA, June 1992.

[34] Richard P. Paul. *Robot Manipulators*. MIT Press, 1981.

[35] Marc H. Raibert and Jessica K. Hodgins. Animation of dynamic legged locomotion. *Computer Graphics*, 25(4):349–358, July 1991.

[36] A. James Stewart and James F. Cremer. Animation of 3D human locomotion: Climbing stairs and descending stairs. In *Eurographics Workshop on Animation and Simulation*, 1992.

[37] Michiel van de Panne and Eugene Fiume. Sensor-actuator networks. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 335–342, August 1993.

[38] Michiel van de Panne, Eugene Fiume, and Zvonko Vranesic. Physically based modeling and control of turning. *Graphical Models and Image Processing*, 55(6):507–521, November 1993.

[39] Nickos Vasilonikolidakis and Gordon J. Clapworthy. Inverse lagrangian dynamics for animating articulated models. *The Journal of Visualization and Computer Animation*, 2:106–113, 1991.

[40] M. Vukobratović. *Biped Locomotion.* Scientific Fundamentals of Robotics 7, Communications and Control Engineering Series. Springer-Verlag, Berlin, New York, 1990.

[41] Kenneth J. Waldron. Force and motion management in legged locomotion. *IEEE Journal of Robotics and Automation*, RA-2(4):214–220, December 1986.

[42] Jane Wilhelms. Using dynamic analysis for realistic animation of articulated bodies. *IEEE Computer Graphics and Applications*, 7(6):12–27, June 1987.

[43] Andrew Witkin and Michael Kass. Spacetime constraints. *Computer Graphics*, 22(4):159–168, August 1988.

[44] Tsuneo Yoshikawa. *Foundations of Robotics.* MIT Press, 1990.

Appendices

A Newton-Euler Method

Some notations must be introduced. Unless otherwise mentioned, all the vectors and matrices are resolved in world coordinates. f_i^J is the force exerted on link i by link $i-1$ at joint i. n_i^J is the moment exerted on link i by link $i-1$ at joint i. f_i^L is the apparent force exerted on link i, considering its positional acceleration. n_i^L is the apparent moment exerted on link i, considering its angular acceleration. f_i^{ext} is the external force exerted on the COM of link i. n_i^{ext} is the external moment exerted on the COM of link i. O_i is the origin of link i. C_i is the COM of link i. θ_i is a scalar quantity that represents the joint angle at joint i. ω_i is the angular velocity of link i. v_i is the positional velocity of O_i. v_{C_i} is the positional velocity of C_i. m_i is the mass of link i. I_i is the inertia tensor of link i. z_i is the unit vector in the direction of the joint axis i.

The outward iteration ($i = 0, \ldots, s-1$) computes the accelerations and the apparent link forces and moments [11]:

$$\begin{aligned}
\omega_{i+1} &= \omega_i + \dot{\theta}_{i+1} z_{i+1} && (6)\\
\dot{\omega}_{i+1} &= \dot{\omega}_i + \omega_i \times \dot{\theta}_{i+1} z_{i+1} + \ddot{\theta}_{i+1} z_{i+1} && (7)\\
\dot{v}_{i+1} &= \dot{v}_i + \dot{\omega}_i \times \overrightarrow{O_iO}_{i+1} + \omega_i \times (\omega_i \times \overrightarrow{O_iO}_{i+1}) && (8)\\
\dot{v}_{C_{i+1}} &= \dot{v}_{i+1} + \dot{\omega}_{i+1} \times \overrightarrow{O_{i+1}C}_{i+1} + \omega_{i+1} \times (\omega_{i+1} \times \overrightarrow{O_{i+1}C}_{i+1}) && (9)\\
f_{i+1}^L &= m_{i+1} \dot{v}_{C_{i+1}} && (10)\\
n_{i+1}^L &= I_{i+1}\dot{\omega}_{i+1} + \omega_{i+1} \times I_{i+1}\omega_{i+1} && (11)
\end{aligned}$$

The inward iteration ($i = s-1, \ldots, 1$) can now begin computing the joint forces and moments. Considering all the forces acting on link i, we have

$$f_i^J - f_{i+1}^J + f_i^{ext} = f_i^L, \qquad (12)$$

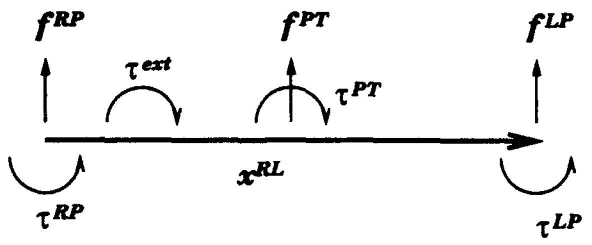


Figure 6: The Pelvis Diagram

which can be solved for f_i^J

$$f_i^J = f_{i+1}^J + f_i^L - f_i^{ext}. \tag{13}$$

Considering all the moments acting on link i,

$$n_i^J - n_{i+1}^J + n_i^{ext} + \vec{O_iO_i} \times f_i^J + \vec{O_iC_i} \times f_i^{ext} + \vec{O_iO}_{i+1} \times (-f_{i+1}^J) = n_i^L \tag{14}$$

which can be solved for n_i^J,

$$n_i^J = n_{i+1}^J - n_i^{ext} - \vec{O_iC_i} \times f_i^{ext} + \vec{O_iO}_{i+1} \times f_{i+1}^J + n_i^L \tag{15}$$

Once n_i^J is computed, the joint torque τ_i is given as

$$\tau_i = n_i^J \cdot z_i \tag{16}$$

It is easy to see that this algorithm is $O(s)$, where s is the number of the links.

B Resolving the Closed Loop at the Lower Limbs

Let f^{PT} and τ^{PT} be the force and torque exerted on upper body by the pelvis (Figure 6). Let f^{LP} and f^{RP} be the reaction forces acting on the pelvis from the left and right thighs, respectively. Let f^{ext} be the external force on the pelvis. Let τ^{LP} and τ^{RP} be the torque exerted from the left and right thighs to the pelvis, respectively. Let τ^{ext} be the external torque acting on the pelvis.

When an apparent force F^P (when the mass and acceleration of the pelvis are m and $\dot{v}^c_{pelvis}$, $F^P = m \cdot \dot{v}^c_{pelvis}$) is acting on the pelvis, we have the following relation between the forces:

$$f^{ext} - f^{PT} + f^{LP} + f^{RP} = F^P \tag{17}$$

Let $f = f^{LP} + f^{RP}$, then

$$f = -f^{ext} + f^{PT} + F^P \tag{18}$$

We distribute f according the the relative distances a_l and a_r of the projection of the center of mass from the ankles. Thus $f^{LP} = a_l \cdot f$ and $f^{RP} = a_r \cdot f$, where $a_l + a_r = 1, a_l \geq 0, a_r \geq 0$.

Let τ_{fTP} be the moment generated by f^{TP}, which is given by $\frac{1}{2}x^{RL} \times f^{TP}$, where x^{RL} is the vector from the right hip to the left hip. Let τ_{fLP} be the moment generated by f^{LP}, which is given by $x^{RL} \times f^{LP}$. Let τ_P be the apparent moment on the pelvis, and τ_{ext} be the external moment on the pelvis. We have

$$\tau^{ext} + \tau^{LP} + \tau^{RP} - \tau^{PT} + \tau_{fTP} + \tau_{fLP} = \tau_P. \tag{19}$$

Similarly as above, τ^{LP} and τ^{RP} are given by $a_l \cdot \tau$ and $a_r \cdot \tau$, where

$$\tau = -\tau^{ext} + \tau^{PT} - \tau_{fTP} - \tau_{fLP} + \tau_P. \tag{20}$$

D Blending and Morphing of Dynamic Shapes: Douglas DeCarlo and Dimitri Metaxas

Blending and Morphing of Dynamic Shapes

Douglas DeCarlo and Dimitri Metaxas

Department of Computer and Information Science,
200 South 33rd St.
University of Pennsylvania, Philadelphia, PA 19104

Category: Research paper, **Format:** Regular paper.

Abstract

This paper develops a new approach to shape modeling based on blending. Blended shape models are constructed by performing a linear interpolation of parameterized shapes using a blending function. This blending function is a B-spline that specifies the way shapes are combined, and has control points that are parameters of the shape. The blended shape can have aspects of each of the shapes of which it is composed. In this way, we can create models that can change genus to be that of any of the component shapes. These shapes use a small number of global parameters, and hence compactly represent complex shapes. Our models also include local deformations, which are added on top of the blended shape, and can be used to create fine detail. Using a physics-based approach, these geometric models are transformed into deformable models that deform based on forces from a simulated physical environment. By devising rules for transforming between blended shapes, we develop a variation of morphing called parameter morphing. These models can also be used for shape estimation applications by having blended models conform to data sets due to forces applied by the data. We present our technique through a series of dynamic animations.

Blending and Morphing of Dynamic Shapes

Abstract

This paper develops a new approach to shape modeling based on blending. Blended shape models are constructed by performing a linear interpolation of parameterized shapes using a blending function. This blending function is a B-spline that specifies the way shapes are combined, and has control points that are parameters of the shape. The blended shape can have aspects of each of the shapes of which it is composed. In this way, we can create models that can change genus to be that of any of the component shapes. These shapes use a small number of global parameters, and hence compactly represent complex shapes. Our models also include local deformations, which are added on top of the blended shape, and can be used to create fine detail. Using a physics-based approach, these geometric models are transformed into deformable models that deform based on forces from a simulated physical environment. By devising rules for transforming between blended shapes, we develop a variation of morphing called parameter morphing. These models can also be used for shape estimation applications by having blended models conform to data sets due to forces applied by the data. We present our technique through a series of dynamic animations.

Category: Research paper
Format: Regular paper
Keywords: Shape and Object Representation, Shape Blending, Parameter Morphing, Deformable Models, Physics-Based Modeling, Shape Estimation, Finite Elements.

1 Introduction

Shape modeling is an important part of computer graphics and solid modeling, and has attracted significant attention in the past decades. Due to the variety of applications, it requires the use of models that maximize shape coverage, represent shape detail, provide a useful abstraction of shape, and are useful in shape morphing, and dynamic animations. Most of the existing shape models and associated techniques are limited to subsets of the above requirements. In this paper we present a unified approach to shape modeling that develops models and techniques that are useful in a wide range of applications. Using these models, the shape can be intuitively and efficiently represented by a few parameters that cover a large number of different shapes including shapes with varying genus. The flexibility of these models comes from the use of a blending function, which is used to combine parameterized primitives.

A class of dynamic locally and globally deformable primitives was introduced in [13, 22]. These primitives include global deformation parameters that represent the salient shape features of natural parts, and local deformation parameters which capture shape details. These models have a geometric structure that allows the combination of parametric models (such as superquadrics), parameterized global deformations (such as bending or twisting) and local spline free-form deformations. In this way, the descriptive power of these models is a superset of locally deformable models [21] and globally deformable models [15, 24].

The main limitation of these deformable primitives, and every other shape model used in computer graphics, is that they are not general enough to represent a very large number of possible shapes that may be needed in an animation with a few intuitive parameters. We move toward overcoming this limitation by developing a new class of deformable models that we dub *Blended Deformable Models*. These new deformable models are a superset of previously defined deformable models and allow genus changes (a sphere can deform into a torus). Our blended shapes are a linear interpolation of any two shapes that can be defined parametrically on a common material coordinate space. The linear interpolation is performed using a blending function that specifies how the two shapes are blended together. For example, a sphere and a cylinder blended together could produce a bullet shaped object (see Fig. 2). A sphere and torus blended together would produce an object that could be either genus 0 or genus 1, depending on the blending function. Fig. 5 shows a variety of shapes that we can model by blending parameterized primitives.

Dynamics is added to these blended models using the previously developed physics-based framework of [13]. The blended models introduced here do not require additional machinery to fit into this framework.

Under normal conditions, it is not topologically possible for the genus of a shape to change. Nevertheless, we present a systematic technique for changing the genus of a model by altering the connectivity of the surface without introducing discontinuities in shape. This alteration occurs at the point when a sphere is dimpled inward at the poles so that it resembles a torus with a zero sized hole (see Fig. 3).

To improve the behavior of the blending and to make the object tessellation more uniform (such as for a "square" superquadric), we permit scaling of the material coordinate space. In conjuction with rules for blending between classes of shapes, this operation makes the models useful for shape morphing. This morphing, refered to as *parameter morphing*, in contrast to traditional graphics morphing applications, is based on parameter interpolation. We also show how

0

blended shapes can be used for shape estimation using the framework of [12].

2 Related Work

Several researchers have proposed shape models that are suitable for shape representation, morphing and shape estimation. A variety of volumetric primitives have been proposed in [1, 2, 4]. The integration of dynamics with these primitives was shown in [16, 21]. Global deformations of objects are presented in [2, 24], and local deformations have been presented in [17]. A hybrid of global and local deformations was presented in [13, 22]. Other shape representations such as [19] used oriented particle systems to represent complex surfaces using connectivity information, but do not provice a compact abstraction of shape. The field of solid modeling has produced powerful shape representation techniques using surface blends [7, 8, 23], but these techniques are not convenient for dynamic animations and shape estimation.

Many shape blending and morphing techniques have been developed [3, 10, 11, 18]. The characteristic of these techniques is the development of algorithms for interpolation of shapes between an initial and a final shape, and to establish a correspondence between points on the shape. None of these techniques are based on the use of model parameters due to the lack so far of parametric models with sufficiently large shape coverage. Recent shape estimation techniques in computer graphics [6, 9] are based on local shape representations only, making it computationally inefficient for shapes that can be described parameterically (e.g., a cube, a torus).

3 Geometry of Deformable Models

In general, our models are 3D shapes whose material coordinates $\mathbf{u} = (u, v)$ are defined in a domain Ω. The positions of points on the model relative to an inertial frame of reference Φ in space are given by a vector-valued, time varying function $\mathbf{x}(\mathbf{u}, t)$. We set up a non-inertial, model-centered reference frame ϕ and express the position function as

$$\mathbf{x} = \mathbf{c} + \mathbf{R}\mathbf{p}, \tag{1}$$

where $\mathbf{c}(t)$ is the origin of ϕ at the center of the model and the rotation matrix $\mathbf{R}(t)$ gives the orientation of ϕ relative to Φ. Thus, $\mathbf{p}(\mathbf{u}, t)$ gives the positions of points on the model relative to the model frame.

We further express

$$\mathbf{p} = \mathbf{s} + \mathbf{d} \tag{2}$$

as the sum of a reference shape $\mathbf{s}(\mathbf{u}, t)$ and a local displacement $\mathbf{d}(\mathbf{u}, t)$.

3.1 Global Deformations

As in [13], the reference shape is defined as

$$\mathbf{s} = \mathbf{T}(\mathbf{e}; b_0, b_1, \ldots), \tag{3}$$

where $\mathbf{e}$ represents a geometric primitive

$$\mathbf{e}(\mathbf{u}; a_0, a_1, \ldots) \tag{4}$$

defined parametrically in $\mathbf{u}$ and parameterized by the variables a_i. The shape represented by $\mathbf{e}$ is subjected to the *global deformation* $\mathbf{T}$ which depends on the global deformation parameters b_i.

Although generally non-linear, $\mathbf{e}$ and $\mathbf{T}$ are assumed to be differentiable so that we may compute the Jacobian of $\mathbf{s}$. $\mathbf{T}$ may be a composite sequence of primitive deformation functions $\mathbf{T}(\mathbf{e}) = \mathbf{T}_1(\mathbf{T}_2(\ldots \mathbf{T}_n(\mathbf{e})))$.

For the examples in this paper, we use a global bending deformation similar to that given by Barr [2] along the z axis in the x direction.

In the next section, the primitive definition for a supertoroid is presented. Note that any parameterized primitives (not just superquadrics and supertoroids) can be used for shape blending. Subsequently, we describe our shape blending technique.

3.1.1 Supertoroid Definition

The definition for the supertoroid primitive (with 7 parameters) is given by

$$\mathbf{e}_{torus} = \begin{pmatrix} \frac{a_1}{a_4+1}(a_4 + C_{2u}{}^{\epsilon_1})C_v{}^{\epsilon_2} \\ \frac{a_2}{a_5+1}(a_5 + C_{2u}{}^{\epsilon_1})S_v{}^{\epsilon_2} \\ a_3 S_{2u}{}^{\epsilon_1} \end{pmatrix} \quad \begin{array}{l} u \in (-\pi/2, \pi/2] \\ v \in (-\pi, \pi] \end{array} \tag{5}$$

$$C_u{}^{\epsilon} = \mathrm{sgn}(\cos u)\cos^{\epsilon}|u| \qquad S_u{}^{\epsilon} = \mathrm{sgn}(\sin u)\sin^{\epsilon}|u|$$

where $a_1, a_2, a_3, \epsilon_1, \epsilon_2 \geq 0$ and $a_4, a_5 \geq 1$. a_1, a_2 and a_3 are size parameters in the x, y and z directions respectively. ϵ_1 and ϵ_2 are squareness parameters as in a superquadric. a_4 and a_5 are hole radius parameters in the x and y directions. The hole is closed for values of 1, and opens as these parameters increase.

This is a similar definition of a supertoroid given by Barr [1]. One difference is the addition of a_5, a second inner radius parameter, which adds flexibility in the shape coverage of this primitive by allowing asymmetric holes.

The second change is the addition of the scaling factors $1/(a_4+1)$ and $1/(a_5+1)$. This separates the effects of the global size parameters (a_1, a_2 and a_3) from the inner radii parameters (a_4 and a_5) to allow hole size changes that do not affect the global torus size.

3.1.2 Blended Models

In a method analogous to the linear interpolation of two points, it is possible to blend two functions. Given two functions, $f(x)$ and $g(x)$, we can blend them using a third function, $\alpha(x)$ (with range $[0, 1]$), so that

$$h(x) = f(x)\alpha(x) + g(x)(1 - \alpha(x)). \tag{6}$$

An example of this is shown in Fig. 1. Notice how $h(x) \approx f(x)$ where $\alpha(x) \approx 1$, $h(x) \approx g(x)$ where $\alpha(x) \approx 0$, and how $h(x)$ is between $f(x)$ and $g(x)$ everywhere.

Using this idea, we can blend parameterized shapes (such as superquadrics) by the following:

$$\mathbf{s}(u, v) = \mathbf{s}_1(u, v)\alpha(u) + \mathbf{s}_2(u, v)(1 - \alpha(u)) \tag{7}$$

where $\mathbf{s}_1$ and $\mathbf{s}_2$ are two shape primitives parameterized over Ω, as in Fig. 2(a) and (b). While it would be possible to have α vary with u and v, we restrict α so it does not vary with v. Greater shape coverage is

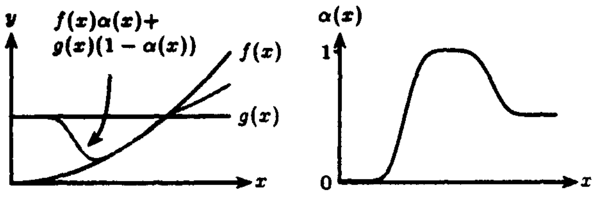


Figure 1: Blending of two functions $f(x)$, $g(x)$ given blending function $\alpha(x)$

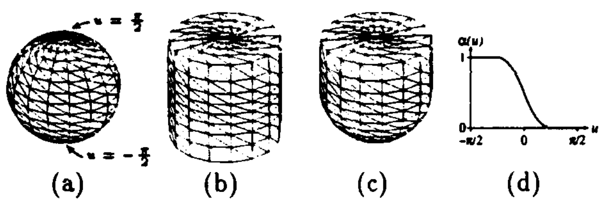


Figure 2: (a) Shape $\mathbf{s}_1$ (b) Shape $\mathbf{s}_2$ (c) Blended shape $\mathbf{s}$ (d) Blending function $\alpha(u)$

achieved by blending several already blended shapes. This produces a more hierarchical structure of shape.

Fig. 2(c) shows the result of blending the shapes shown in Fig. 2(a) and (b). The blending function used to blend the shapes is shown in Fig. 2(d). The blending is performed along u, which corresponds to the z-axis in these shapes (from pole to pole). Notice how the "top" of $\mathbf{s}$ looks like $\mathbf{s}_2$ since $\alpha(\frac{\pi}{2}) = 0$, and how the "bottom" of $\mathbf{s}$ looks like $\mathbf{s}_1$ since $\alpha(-\frac{\pi}{2}) = 1$.

The global parameters of $\mathbf{s}$ will include the global parameters of $\mathbf{s}_1$ and those of $\mathbf{s}_2$. It will also include any global deformation parameters in $\mathbf{T}$. Note that while $\mathbf{T}$ is applied separately to each shape, there is only one set of global deformation parameters for the blended shape. The parameters to describe α are also included (see section 3.1.5).

Any two parametrically defined shapes that have the same domain Ω can be blended this way (this includes other blended shapes). If the closure of both domains is the same, the blending can still be applied directly, assuming the shape "wraps around" to the open area in Ω space. If they are not defined over the same domain, an invertible mapping can be found to make the domains agree (after closing both domains).

When blending shapes, not all combinations of shapes will achieve the desired result. For example, if two shapes are incompatible, the blended result may be non-intuitively shaped and interpenetrating. There must also be a correspondence between the two shapes so that the points that should be blended are at the same position in Ω space. Section 6.1 shows how this correspondence can be altered without changing the shape to make two shape parameterizations compatible.

3.1.3 Genus changing

It is also possible to blend objects of different genus, such as a sphere (genus 0) and a torus (genus 1). The resulting blended object will have a hole which can appear depending on α.

Obviously, there is no smooth transition between these two shapes because they are not homeomorphic. Yet it is possible to have a transition between the two where there is a single discontinuous event – when the object changes genus. This event only affects the topology of the object, and does not have an effect on shape. The transformation is shown in Fig. 3.

In Fig. 3, $\mathbf{s}_1$ is a torus, and $\mathbf{s}_2$ is a sphere. Initially, $\alpha(u) = 0$, and the genus of the shape is 0. We compute the blended result $\mathbf{s}$ using (7).

Fig. 3(a) shows the initial sphere. After this, the nodes slowly dimple inward (b) until they touch (c). The values of $\alpha(\frac{\pi}{2})$ and $\alpha(-\frac{\pi}{2})$ change from 0 to 1 to produce this dimpling. The object is topologically a sphere, but in (c), has the same shape as a torus with a zero size hole (with $a_4 = a_5 = 1$).

It is at this point where the topology changes to be that of a torus (c). This point is easily detected by observing the blending function at its endpoints. The connectivity of the surface is changed to accommodate this change in topology. A discussion of how the node interconnections change is given in section 3.1.4. The hole can now be opened by increasing the torus inner radius parameters, shown in (d) and (e) (which are shown from a different viewpoint to make the hole visible).

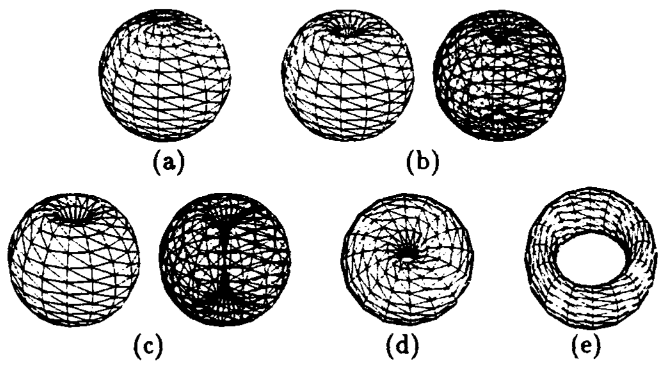


Figure 3: Changing from a sphere (a) to a torus (e)

There are two constraints on the parameters of a torus-sphere blend that must be enforced to insure the blended shape remains closed. Whenever the object is genus 0, the torus parameters a_4 and a_5 must be 1 (to form a zero size hole in the torus). The second restriction is whenever the object has genus 1, $\alpha(\frac{\pi}{2})$ and $\alpha(-\frac{\pi}{2})$ must have the value 1 (intermediate values of α need not be constrained). This way, the blended object is completely toroidal at the center of the hole.

This idea for changing the genus of shapes can be applied to any shape primitives. In this case, since α varies only with u, the types of holes that can be added is limited.

3.1.4 Node Interconnections

When changing the genus of an object, the mesh of nodes must be reconnected to conform to the new topology. This is a straightforward, but necessary part of the genus conversion process. Fig. 4 shows how Ω is "folded up" to produce a sphere or torus. The arrows in these diagrams indicate two nodes being "merged" together, since the material coordinates of the nodes map to the same model coordinates. For both the sphere and the torus, a tube is made first (the dotted lines). For a sphere (a), the north and south poles are created by closing each end of the tube. For a torus (b), the ends of the tube are connected together.

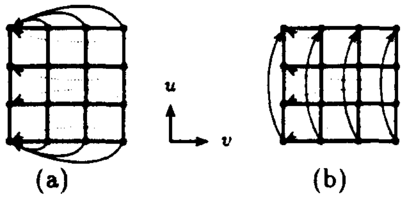


Figure 4: Node interconnection differences between a sphere (a) and torus (b)

When the genus changes, the node mesh must be unfolded, and re-folded to have the proper configuration. Elements must also be either added or removed to eliminate any degenerate elements.

3.1.5 Blending Function Parameterization

The blending function is implemented as a piecewise non-uniform C^1 quadratic B-spline [5]. These curves are used because they can easily be made into x-y functions, and can be bounded to a particular range using their convex hull property. Their endpoint interpolation property also proves to be useful to enforce the constraint on the blending function α when the genus changes.

Given different types of shape primitives, the domain of α may change. To generalize the implementation, we will define a canonical blending function, $\beta : [0,1] \rightarrow [0,1]$.

For the case of a superquadric, we use the blending function $\alpha : [-\frac{\pi}{2}, \frac{\pi}{2}] \rightarrow [0,1]$. We can obtain β by scaling the domain so that we have

$$\beta\left(\frac{u+\frac{\pi}{2}}{\pi}\right) = \alpha(u). \tag{8}$$

A C^1 quadratic B-spline of L pieces has $2L+1$ control points, $\{\mathbf{b}_i = (b_{x_i}, b_{y_i}) \mid i \in 0 \ldots 2L\}$, and $L+1$ knots, $\{u_i \mid i \in 0 \ldots L\}$ (with $u_i \leq u_j$ for all $0 \leq i \leq j \leq L$). Of the $2L+1$ control points, $L+2$ points (called inner control points), are needed to specify the curve. The inner points of a B-spline are $\mathbf{b}_0$, $\{\mathbf{b}_{2i+1} \mid i \in 0 \ldots L-1\}$, and $\mathbf{b}_{2L}$.

The remaining $L-1$ points (called junction points) are added to enforce C^1 continuity, and are easily computed from the inner points and knots [5]. There is one junction point for each knot except u_0 and u_L. The curve passes through each junction point, and the endpoints.

The de Casteljau algorithm [5] is used to find values of β at a particular point for each curve segment. This results in a piecewise parametric function in t, $\mathbf{p}(t) = (x\ y)^{\mathrm{T}}$ with $t \in [u_0, u_L]$. To insure $\mathbf{p}(t)$ is a function with domain $[0,1]$, we set $x = t$, $u_0 = 0$ and $u_L = 1$. To find the x-components of the control points, we solve $\mathbf{p}(t) = (x\ y)^{\mathrm{T}} = (t\ y)^{\mathrm{T}}$, resulting in

$$b_{x_i} = \begin{cases} 0 & i = 0 \\ \frac{u_{(i-1)/2} + u_{(i+1)/2}}{2} & \text{odd } i \\ 1 & i = 2L \\ u_{i/2} & \text{otherwise (junction points)} \end{cases} \tag{9}$$

This construction yields a C^1 quadratic B-spline function, which can become C^0 if two knots have equal value. The curve is interpolated exactly at the endpoints and at each junction point, and is guaranteed to always be in the range $[0,1]$.

3.1.6 Shape coverage

By using different blending functions, we can create a variety of shapes shown in Fig. 5. The first four shapes are combinations of two superquadrics. Notice how some of the objects have both "square" and "round" areas, which is not easily representable using tapering or other global deformations. The last two shapes are a combination of a superquadric and a supertoroid.

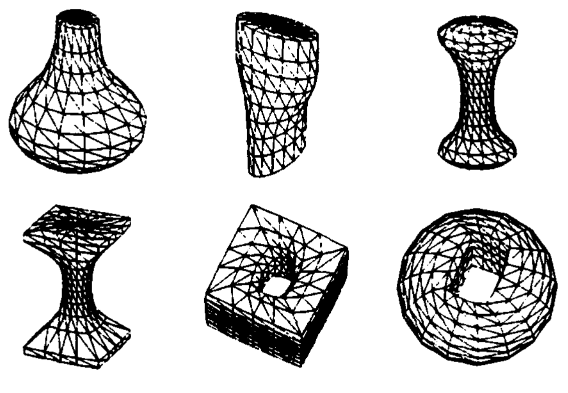


Figure 5: Examples of blended shapes using superquadrics and supertoroids (without any global parameterized deformations)

3.2 Local Deformations

The implementation of local deformations uses C^1 finite elements to represent the local displacements, as in [22, 12]. This method does not require any alteration for shape blending applications.

We use model that simulates a thin plate under tension deformation energy, suitable for a C^1 model surface [12]. The strain energy is approximated using triangular finite elements [13] whose shape functions are tensor products of one-dimensional Hermite polynomials [25].

4 Dynamics

The dynamics framework in [13], requires the computation of the shape model jacobian, $\mathbf{J}$. $\mathbf{q} = (\mathbf{q}_c^{\mathrm{T}}, \mathbf{q}_\theta^{\mathrm{T}}, \mathbf{q}_s^{\mathrm{T}}, \mathbf{q}_d^{\mathrm{T}})^{\mathrm{T}}$ is the vector of the model's generalized coordinates , where $\mathbf{q}_c = \mathbf{c}$, $\mathbf{q}_\theta = \boldsymbol{\theta}$, and $\mathbf{q}_s = (\mathbf{q}_{s_1}^{\mathrm{T}}, \mathbf{q}_{s_2}^{\mathrm{T}}, \mathbf{q}_b^{\mathrm{T}}, \mathbf{q}_T^{\mathrm{T}})^{\mathrm{T}}$ ($\mathbf{q}_{s_1}$ and $\mathbf{q}_{s_2}$ are the parameters of each of the blended shapes, $\mathbf{q}_b$ is described in section 4.1, and $\mathbf{q}_T$ are the parameters of the parameterized global deformations, such as bending). $\mathbf{q}_o$ are the "rest" parameters of the shape, analogous to the rest length of a Hookean spring.

To specify the dynamics, we introduce a mass distribution $\mu(\mathbf{u})$ over the model and assume that the material is subject to frictional damping. We also assume that the material may deform elastically or viscoelastically.

From Lagrangian mechanics we obtain second-order equations of motion which take the form

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{D}\dot{\mathbf{q}} + \mathbf{K}(\mathbf{q} - \mathbf{q}_o) = \mathbf{f}_q + \mathbf{g}_q, \qquad (10)$$

where $\mathbf{M}$, $\mathbf{D}$ and $\mathbf{K}$ are the generalized mass, damping and stiffness matrices, repsectively. $\mathbf{f}_q$ is the vector of generalized external forces applied to the model (such as gravity, friction and collision forces). $\mathbf{g}_q$ is the vector of generalized inertial forces, which include coriolis and centrifugal forces. The derivation and computation of these quantities is described in detail in [12].

4.1 Blending function dynamics

The parameters used to construct the blending function are the $L+2$ control point b_y values and the $L-1$ movable knots (u_0 and u_L are fixed), which is $2L+1$ total parameters to specify α. We concatenate all these parameters into the vector $\mathbf{q}_b$, so that

$$\begin{aligned}\mathbf{q}_b \quad = \quad & (b_{y_0}, b_{y_1}, \ldots, b_{y_{2i+1}}, \ldots, b_{y_{2L-1}}, b_{y_{2L}}, \\ & u_1, \ldots, u_{L-1})^{\mathsf{T}}. \qquad (11)\end{aligned}$$

To determine how these parameters change due to applied forces, the jacobian is needed. Given (7) above,

$$\mathbf{J}_b = \frac{\partial \mathbf{s}(u,v)}{\partial \mathbf{q}_b} = (\mathbf{s}_1(u,v) - \mathbf{s}_2(u,v))\,\frac{\partial \alpha(u)}{\partial \mathbf{q}_b} \qquad (12)$$

To compute $\frac{\partial \alpha(u)}{\partial \mathbf{q}_b}$, we use a simple modification to the de Casteljau algorithm using the product rule. To make sure the components of $\mathbf{q}_b$ have reasonable values, the constraints $b_y \in [0,1]$ and $u_i \leq u_j$ for all $0 \leq i \leq j \leq L$ are enforced.

The addition of blending also changes the jacobian for the global shape. $\mathbf{J}_{s_1}$ is the jacobian for the first shape (with respect to $\mathbf{q}_{s_1}$), and $\mathbf{J}_{s_2}$ is the jacobian for the second shape (with respect to $\mathbf{q}_{s_2}$). The jacobian for the blended shape using (7)

$$\mathbf{J} = \left(\alpha(u)\mathbf{J}_{s_1}^{\mathsf{T}}, (1-\alpha(u))\mathbf{J}_{s_2}^{\mathsf{T}}, \mathbf{J}_b^{\mathsf{T}}\right)^{\mathsf{T}} \qquad (13)$$

Intuitively, this means the jacobians for the components of a blended shape have a greater or lesser effect at a particular location depending on the function α.

5 Parameter Morphing

Parameter morphing is a different approach to morphing than image or volume morphing [3, 10, 11] and 2-D shape blending [18]. Much of the difficulty in these methods is establishing a correspondence between the two objects being morphed.

When morphing between two parameterized shapes, a correspondence has already been established using the material coordinate space. As a result, by simply changing the parameters of the object from one set to another, a reasonable metamorphosis is obtained.

The morphing between two parameterized shapes can be performed by a simple linear interpolation of their parameters across time. For now, we will assume that the two shape models to be morphed have the same structure (they are composed of the same blended primitives, and have the same number of blending parameters). Section 5.1 describes how two differing shape models can be made compatible for morphing purposes.

When morphing between two parameterized shapes (which have the same parameterization), we have the shapes $\mathbf{s}$ and $\mathbf{s}'$, which have parameters $\mathbf{q}$ and $\mathbf{q}'$ respectively. The duration of the morph is t_m. We will denote the elapsed time of the morph as t_e, where $0 \leq t_e \leq t_m$.

For morphing where there is no change of genus, all parameters are interpolated directly, so that for the parameters in $\mathbf{q}_s$ and $\mathbf{q}_d$,

$$\mathbf{q}_{\mathrm{morph}_i} = \mathbf{q}_i\left(1 - \frac{t_e}{t_m}\right) + \mathbf{q}_i'\left(\frac{t_e}{t_m}\right) \qquad (14)$$

For morphing instances where there is a genus change, the morphing must be broken down into two non-overlapping stages. The first stage is to place the object in a state where it is legal to change genus. The second stage would be to open the hole. During each of these stages, those parameters that are not restricted by genus are interpolated using (14).

For the case of morphing from a sphere to a torus using a sphere-torus blended model, the first stage has a_4 and a_5 of the torus fixed, and the blending function endpoints morphing to the torus side. At the end of this stage, the sphere-torus will be in a state where the hole can be opened. The second stage has a_4 and a_5 morphing to their final values. This produces a smooth transition during a genus change. In order for these parameters to change by the end of their stages, for the affected parameters, t_e is measured from the start of the stage, and t_m is the stage duration. Morphing from a torus to a sphere is accomplished by performed the above in the reverse order.

5.1 Morphing and Blending

When morphing between two objects that have different shape models, we create a blended type that blends the two models, and morphs between them. For example, to morph between a torus and a sphere, create both objects as a torus-sphere blend, and morph the blended shapes. In order to have intuitive intermediate shapes, both parts of each blended shape should have a similar appearance. For example, if a torus is "upgraded" to a sphere-torus blend, the sphere size parameters (a_1, a_2 and a_3) should be set to those of the torus.

Knot insertion methods [5] can be used to insert additional control points of the blending function to morph two models with different numbers of blending parameters.

5.2 Morphing and Dynamics

When an object is morphing, the velocities of nodes are affected. This can affect the computation of coriolis forces (part of $\mathbf{g}_q$) and friction forces. To integrate morphing with dynamics to affect these velocities, an additional term to $\dot{\mathbf{q}}_s$ and $\dot{\mathbf{q}}_d$ is added when computing nodal velocities, $\dot{\mathbf{p}}$, so that

$$\dot{\mathbf{p}} = \dot{\mathbf{s}} + \dot{\mathbf{d}} = \mathbf{J}(\dot{\mathbf{q}}_s + \dot{\mathbf{q}}_{s_{\mathrm{morph}}}) + \mathbf{S}(\dot{\mathbf{q}}_d + \dot{\mathbf{q}}_{d_{\mathrm{morph}}}) \quad (15)$$

where $\dot{\mathbf{q}}_{s_{\mathrm{morph}}}$ and $\dot{\mathbf{q}}_{d_{\mathrm{morph}}}$ are the rates of change of parameters caused by morphing, and are given by

$$\dot{\mathbf{q}}_{s_{morph}} = \mathbf{q}'_s - \mathbf{q}_s \qquad \dot{\mathbf{q}}_{d_{morph}} = \mathbf{q}'_d - \mathbf{q}_d \quad (16)$$

There will also be additional constant factors in the above formulas for those parameters that are changed faster during genus changing stages.

In a dynamic system, it is possible for some of the parameters to change due to forces. When morphing, it would not be correct to simply alter the parameters using (14). Instead, the morphing velocities in (16) can be added to the true velocities inside the numerical integration process. For instance, for an Euler integration with time step Δt, the parameter update for $\mathbf{q}_s$ would be

$$\mathbf{q}_{s_{\mathrm{new}}} = \mathbf{q}_s + \Delta t(\dot{\mathbf{q}}_s + \dot{\mathbf{q}}_{s_{\mathrm{morph}}}) \quad (17)$$

and similarly for $\mathbf{q}_d$.

(16) is used to update the rest shape parameters, $\mathbf{q}_o$. This way, the underlying rest shape is morphed, and the shape will return to this rest shape when no forces are applied. The actual shape has offsets added using (17) to cause the changes in shape due to morphing.

For more interesting effects, substance attributes such as total mass or stiffness values w_0 and w_1 can be morphing using a linear interpolation similar to that in (14).

Morphing is not a real physical process, and can add or remove energy from a system. When objects morph during a collision, the object can "push off" the colliding object if it is becoming larger due to morphing. If an object is becoming smaller due to morphing during a collision, a "cushioning" of the collision is observed. Inertial forces can also be created due to morphing. Objects can lunge a particular direction when the morphing causes the mass of part of an object to shift. Changes in the moment of inertia of an object can affect angular velocity.

6 Model Discretization

6.1 Space scaling

For a shape $\mathbf{e}(\mathbf{u})$ with $\mathbf{u} \in \Omega$, we can scale (or warp) the domain so that we have $\mathbf{e}(\mathcal{S}(\mathbf{u}))$, where $\mathcal{S} : \Omega \rightarrow \Omega$ is invertible. This operation does not change the shape, but instead changes how the material parameters are mapped onto the object (e.g.– for a discretized object, the nodes will change position, but the shape will remain the same).

There are two reasons why this space scaling is necessary. The first reason is to increase the uniformity of the discretization of a shape primitive. With a very uneven discretization, more nodes are needed to compensate, and numerical inaccuracies are introduced. The second reason to scale space is to "line up" two parameterizations to insure the blending has the desired result, and to insure the jacobians of the blending parameters have meaningful values.

The jacobian of $\mathbf{e}$ may have an additional term if any of the parameters of $\mathbf{e}$ are used in $\mathcal{S}$. However, the change associated with this derivative term is not useful for modeling purposes because it originates not from a change in shape, but instead in a change in material parameter space scale.

When discretizing a superquadric, the nodes tend to collect near the "squared" edges if ϵ_1 or ϵ_2 is near zero. Fig. 6(a) shows an example of this uneven node spacing.

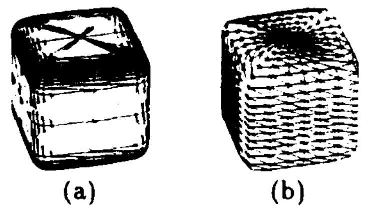


Figure 6: Example of a superquadric ($\epsilon_1 = \epsilon_2 = \frac{1}{4}$) with unscaled (a) and scaled space (b)

Instead of this, we use an approximation by projecting a unit superquadric onto the unit sphere when $\epsilon_1 < 1$ or $\epsilon_2 < 1$, and place nodes on the superquadric where they project down to nodes on the sphere, as in Fig. 7(a). While this solution is approximate, it is fast, and produces adequate results.

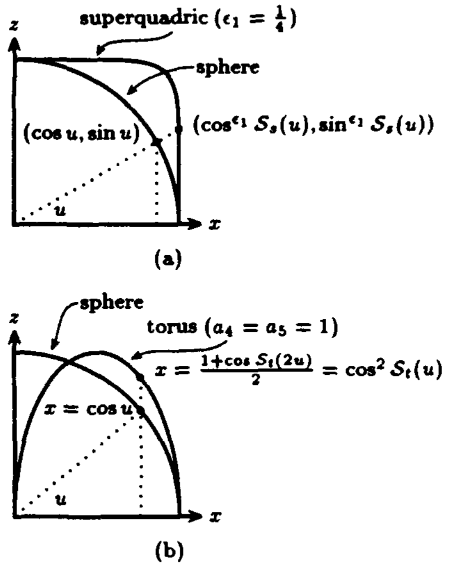

z
superquadric ($\epsilon_1 = \frac{1}{4}$)
sphere
(cos u, sin u)
($\cos^{\epsilon_1} \mathcal{S}_s(u), \sin^{\epsilon_1} \mathcal{S}_s(u)$)
u
x
(a)
z
sphere
torus ($a_4 = a_5 = 1$)
$x = \frac{1+\cos \mathcal{S}_t(2u)}{2} = \cos^2 \mathcal{S}_t(u)$
x = cos u
u
x
(b)

Figure 7: Space scaling for a superquadric (a) and torus (b) (first quadrant, $y = 0$)

In order to find the scaling function $\mathcal{S}_s(u)$, we solve $\tan u = \tan^{\epsilon_1} \mathcal{S}_s(u)$. When we generalize to all quadrants, and do a similar calculation for v and ϵ_2, we obtain

$$\begin{aligned} \mathcal{S}_{s_u}(u) &= \tan^{-1} T_u^{\frac{1}{\epsilon_1}} \quad (18) \\ \mathcal{S}_{s_v}(v) &= \tan^{-1} T_v^{\frac{1}{\epsilon_2}} \end{aligned}$$

where $T_u^\epsilon = \text{sgn}(\tan u)\tan^\epsilon |u|$. This scaling is performed only when $\epsilon_1 < 1$ or $\epsilon_2 < 1$.

Fig. 8 shows the behavior of (18) on u-v space with decreasing values of ϵ. The example where $\epsilon = 0.25$ shows the space scale used to correct the node distribution for the superquadric in Fig. 6(b). The scaling in (18) can also be applied to a supertoroid to correctly rescale any uneven discretization problems caused by torus squareness when $\epsilon_1 < 1$ or $\epsilon_2 < 1$.

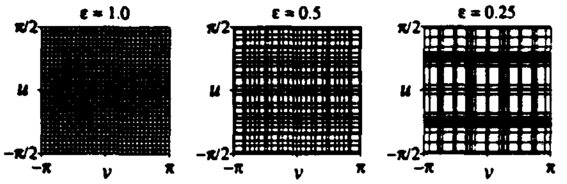


Figure 8: Example of scaled space (superquadric) with varying $\epsilon = \epsilon_1 = \epsilon_2$

The scaling function for a torus ($\mathcal{S}_t$) is used to minimize the effect of the difference in parameterizations of the supertoroid and superquadric when blended. The difference the scaling has on the shape is small, and only makes a significant difference when calculating the jacobian of the blending parameters near the torus hole. If not scaled properly, these jacobians may be nearly parallel to the surface (when they should be nearly perpendicular).

To minimize this difference when a torus and superquadric are blended, we project the unit torus (with a zero size hole) onto the unit sphere, in a direction parallel to the z axis (along the hole) as shown in Fig. 7(b). This projection is performed assuming the torus has a zero sized hole ($a_4 = a_5 = 1$). It is for this case that the jacobians of the blending parameters near the hole are affected most. Solving for $\mathcal{S}_t$, and generalizing for all quadrants, we obtain

$$\begin{aligned} \mathcal{S}_{t_u}(u) &= \text{sgn}\, u \cdot \cos^{-1}\sqrt{\cos u} \\ \mathcal{S}_{t_v}(v) &= v \end{aligned} \qquad (19)$$

When applying more than one space scaling function, those that make parameterizations compatible should be applied first, followed by those that improve tessellations. For the supertoroid case, we would have $\mathbf{e}_{torus}(\mathcal{S}_s(\mathcal{S}_t(u))))$.

6.2 Nodal Mass Computation

For dynamics purposes, the entire mass of the object is assumed to be lumped at the nodes of the object. To produce an accurate simulation, these mass values should reflect the actual mass of the object in the region of the node.

For an evenly spaced tesselation, the mass of each node would be equal. While the methods for altering tesselations described in the previous section improve uneven node distribution, they do not yield evenly spaced tesselations. In general, it is difficult and computationally expensive to produce an evenly spaced tesselation.

We approximate the lumped node mass for node i by computing the ratio of the surface area near node i to the total surface area. We denote A_j as the surface area of element j, where $j \in E$ (E is the index set for elements). $Adj_i \subset E$ is the set of elements that contain node i, as shown in figure 9.

The mass for node i is given by

$$\text{mass}_i = \mu_i \left(\frac{\sum_{j \in Adj_i} A_j}{3 \sum_{j \in E} A_j} \right) \qquad (20)$$

where μ_i is the mass density of the material at node i. The three in the denominator originates from the triangular elements, since the area of each element is divided equally between the nodes it contains. The sum of these nodal masses is the total mass of the material.

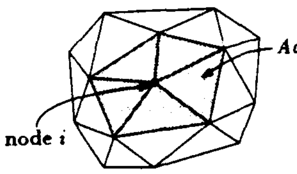


Figure 9: Lumped node mass calculation

7 Shape Estimation

The shape estimation techniques in the framework of [12] can be applied directly to blended shape models.

The order in which parameters are allowed to change is scheduled, so that some parameters remain fixed. For example, the initial fit for a superquadric would vary only a_1, a_2 and a_3. This prevents the shape from being caught in a local minimum energy solution.

To allow genus changing during shape estimation, the constraints on parameters are enforced (such as not allowing the torus hole to open until the topology of the mesh is toroidal). The mesh topology can be altered as described in section 3.1.3. The time of this change is determined by observing the control points of the blending function.

8 Implementation

We demonstrate our technique through a series of experiments involving morphing animations, dynamic animations and shape estimation using our primitives. By using an adaptive Euler technique, our method runs at interactive rates on standard Silicon Graphics workstations. When rendering texture mapped objects, the texture for both objects is drawn using an alpha-transparency blended combination of the textures of both objects. All rendering (including shadows and texture mapping) was performed using the standard SGI GL library.

9 Experiments

Two experiments to demonstrate blended shapes with morphing and dynamics have been performed, and are included on the accompanying animation.

The first experiment exhibits the "pushing off" and "cushioning" that morphing can cause during a collision. The first morph shows the shape getting larger as it collides, so that it jumps higher than it started. The object becomes smaller during second morph, causing the collision to be cushioned. Later, when a similar simulation is run with the timing of the second morph slightly changed, the effect of the "cushioning" is greatly decreased.

The second experiment shows a longer sequence of morphing and dynamic interaction. In the morphing sequence, initially, an apple is at rest on a table. It morphs into a banana, and the bending and uneven mass distribution of the banana causes the object to "push off" the table and start to rotate. Its rotation accelerates as it morphs into an orange, which has a smaller radius than the banana. The orange becomes a mushroom, which has a lower mass than the other objects, and then the apple returns. Next the apple morphs to a donut, demonstrating genus change. After coming to rest on the table again, the object morphs into an orange (at which point the hole closes), then a banana. Again, the bending of the banana gives rise to an inertial force: the banana rotates quickly and "lunges" toward the table. Finally, the object, bouncing on the table, morphs into a mushroom and comes to rest. During this final morph, coriolis forces can be observed, caused by the change in shape during rotation. In the dynamic interaction, we show how to build the apple using a user interface for the construction of blended shapes.

Last is a shape estimation example using incomplete range data of a torus-shaped object. A superquadric-supertoroid model is used and is initialized as a sphere shape. The shape and genus of the model changes dynamically to fit the data.

10 Conclusion

In this paper we developed a new approach to shape modeling by blending parameterized primitives using a B-spline as a blending function. We were able to create a large number of shapes with varying genus using a small number of parameters. By applying physics-based techniques we transformed these models to dynamic models that were used for dynamic animations. Due to their large shape coverage, we used these new models in morphing animations which were based on systematic changes of their parameters. Finally we demonstrated the usefulness of our models in shape estimation from incomplete sparse range data even when a genus change was required during the model fitting process.

References

[1] A. Barr. "Superquadrics and Angle-Preserving Transformations", *IEEE Computer Graphics and Applications*, 1(1), pp. 11–23, 1981.

[2] A. Barr. "Global and Local Deformations of Solid Primitives", *Computer Graphics*, 18, pp. 21–30, 1984.

[3] T. Beier and S. Neely. "Feature-Based Image Metamorphosis", *Computer Graphics (Proc. SIGGRAPH)*, pp. 35-42, 1993.

[4] T. Binford. "Visual Perception by Computer", *IEEE Conference on Systems and Control*, Dec. 1971.

[5] G. Farin. "Curves and Surfaces for Computer Aided Geometric Design", Academic Press, Second edition, 1990.

[6] M. Halstead, M., Kass, M., and DeRose, T. "Efficient, Fair Interpolation using Catmull-Clark Surfaces", *Computer Graphics (Proc. SIGGRAPH)*, pp. 35-44, 1993.

[7] C.M. Hoffmann and J. Hopcroft. "The Geometry of Projective Blending Surfaces", *Artificial Intelligence*, 37, pp. 357–376, 1988.

[8] C.M. Hoffmann. "Geometric and Solid Modeling", Morgan-Kaufmann, Palo Alto, 1989.

[9] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle. "Surface Reconstruction from Unorganized Points", *Computer Graphics (Proc. SIGGRAPH)*, pp. 71-75, 1992.

[10] J. Hughes. "Scheduled Fourier Volume Morphing", *Computer Graphics (Proc. SIGGRAPH)*, pp. 43-46, 1992.

[11] J. Kent, W. Carlson, R. Parent. "Shape Transformation for Polyhedral Objects", *Computer Graphics (Proc. SIGGRAPH)*, pp. 47-54, 1992.

[12] D. Metaxas. "Physics-Based Modeling of Nonrigid Objects for Vision and Graphics", Ph.D. Thesis, Department of Computer Science, University of Toronto, 1992.

[13] D. Metaxas and D. Terzopoulos. "Dynamic Deformation of Solid Primitives with Constraints", *Computer Graphics (Proc. SIGGRAPH)*, 26(2), pp. 309–312, 1992.

[14] A. Pentland. "Perceptual Organization and the Representation of Natural Form", *Artificial Intelligence*, 28, pp. 293–331, 1986.

[15] A. Pentland and S. Sclaroff. "Closed-Form Solutions for Physically Based Shape Modeling and Recognition", *IEEE Pattern Analysis and Machine Intelligence*, 13(7), pp. 715–729, 1991.

[16] S. Sclaroff and A. Pentland. "Generalized Implicit Functions for Computer Graphics", *Computer Graphics (Proc. SIGGRAPH)*, 25(4):247–250, 1991.

[17] T.W. Sederberg and S.R. Parry. "Free-form Deformation of Solid Geometric Primitives", *Computer Graphics (Proc. SIGGRAPH)*, 20(4), pp. 151–160, 1986.

[18] T. Sederberg and E. Greenwood. "A Physically Based Approach to 2-D Shape Blending", *Computer Graphics (Proc. SIGGRAPH)*, 26(2), pp. 25–34, 1992.

[19] R. Szeliski, D. Tonneson "Surface Modeling with Oriented Particle Systems", *Computer Graphics (Proc. SIGGRAPH)*, 26(2), pp. 185-194, 1992.

[20] R. Szeliski, D. Tonnesen and D. Terzopoulos. "Modeling Surfaces of Arbitrary Topology with Dynamic Particles", *Proc. CVPR '93*, pp. 82-87, New York, June 1993.

[21] D. Terzopoulos, A. Witkin, and M. Kass. "Constraints on Deformable Models: Recovering 3D Shape and Nonrigid motion", *Artificial Intelligence*, 36(1):91–123, 1988.

[22] D. Terzopoulos and D. Metaxas. "Dynamic 3D Models with Local and Global Deformations: Deformable Superquadrics", *IEEE Trans. Pattern Analysis and Machine Intelligence*, 13(7), pp. 703–714, 1991.

[23] J. Warren. "Blending Algebraic Surfaces", *ACM Transactions on Graphics*, 8(4), pp. 263–278, October 1989.

[24] A. Witkin and W. Welch. "Fast Animation and Control of Nonrigid Structures", *Computer Graphics (Proc. SIGGRAPH)*, 24(4), pp. 243–252, 1990.

[25] O. Zienkiewicz. "The Finite Element Method", McGraw-Hill, 1977.

E ANIMATED CONVERSATION: Rule-based Generation of Facial Expression, Gesture and Spoken Intonation for Multiple Conversational Agents: Justine Cassell, Catherine Pelachaud, Norman I. Badler, Mark Steedman, Brett Achorn, Tripp Becket, Brett Douville, Scott Prevost, Chin Seah, and Matthew Stone

ANIMATED CONVERSATION: Rule-based Generation of Facial Expression, Gesture and Spoken Intonation for Multiple Conversational Agents

Justine Cassell Catherine Pelachaud Norman Badler Mark Steedman
Brett Achorn Tripp Becket Brett Douville Scott Prevost Chin Seah Matthew Stone

Department of Computer and Information Science,
200 South 33rd St.
University of Pennsylvania, Philadelphia, PA 19104

Category: Research paper, **Format:** Regular paper.

Abstract

Realistic animation of human-like characters should be based on cognitive behavioral principles to maximize expressive realism. We describe the principles underlying movements performed during conversations between two people. Using these principles, we designed and implemented a system to *automatically* animate conversations between multiple human-like agents with appropriate and synchronized speech, intonation, facial expressions, and hand gestures. The conversation is created by a dialogue planner that produces the text as well as the intonation of the utterances. The speaker/listener relationship, the text, and the intonation in turn drive facial expressions, lip motions, eye gaze, head motion, and arm gesture generators. The facial motion models are driven and synchronized by parallel transition networks. Coordinated arm, wrist, and hand motions are invoked to create semantically meaningful gestures. Throughout we will use examples from an actual synthesized conversation which is fully animated on the accompanying videotape.

1 Introduction

When faced with the task of bringing to life a human-like character, ar ors currently enjoy few options. Either they can manually and laboriously manipulate the n ous degrees of freedom in a synthetic figure, they can write or acquire increasingly sophistic e motion generation software such as inverse kinematics and dynamics, or they can resort to " rformance-based" motions obtained from a live actor. The emergence of low-cost, real-time n on sensing devices has led to renewed interest in active motion capture since 3D position a ientation trajectories may be acquired directly rather than from tedious image rotoscop 36]. Both facial and gestural motions are efficiently tracked from a suitably harnessed act ut this does not imply that the end of manual or synthesized animation is near. Instead i s the challenge of providing a sophisticated toolkit for human character animation that e not require the presence nor skill of a live actor [2]. There are four reasons for this:

- Generalizing sensed motion from a real actor to a synthetic actor of a i rent body size or shape may be difficult.
- The motion sensing hardware may not be available.
- Skilled actor(s) may not be available.
- Several interacting characters may be needed simultaneously (for examp iring a conversation).
- The animator might be unable to manually synthesize convincing motion particular, a program might need to automatically generate a character's gestures and fa l motions during speech or conversation that happens in real-time or in response to a questions or statements.

We believe that the last two points are sufficiently important to justify a continu earch for automatic software synthesis of human motions. These two conditions motivate o stem for ***automatically animating conversations between multiple human-like agents with ap ate and synchronized speech, intonation, facial expressions, and hand gestures.*** Especially worthy is the linkage between speech and gesture which has not been explored before in esizing realistic animation. In people, speech, facial expressions, and gestures are physiolog inked. While an expert animator may realize this unconsciously in the "look" of a proper nated character, a program to automatically generate motions must know the rules in ad This paper presents a working system to realize interacting animated agents.

Conversation is an interactive dialogue between two agents. Conversation includ oken language (words and contextually appropriate intonation marking topic and focus), fac ovements (lip shapes, emotions, gaze direction, head motion), and hand gestures (ha pes, points, beats, and motions representing the topic of accompanying speech). Without al hese verbal and non-verbal behaviors, one cannot have realistic or at least believable aut ous agents. To limit the problems (such as voice and face recognition) that arise from the vement of real human conversants, and to constrain the dialogue, we present the work in rm of a dialogue generation program in which two copies of an identical program having nt knowledge of the world must cooperate to accomplish a goal. Both agents of the conv on collaborate via the dialogue to develop a simple plan of action. They interact with each er to exchange information and ask questions.

In this paper, we first present the background information necessary to establish the s chrony of speech, facial expression, and gesture. We then look at the relevant computer gra literature on face and gesture animation. We can then proceed to a discussion of the system architecture and its several subcomponents. Throughout we will use examples from an actual synthesized conversation which is fully animated on the accompanying videotape.

2 Background

Faces change expressions continuously, and many of these changes are synchronized to what is going on in concurrent conversation. Facial expressions are linked to the content of speech (scrunching one's nose when talking about something unpleasant), emotion (wrinkling one's eyebrows with worry), personality (frowning all the time), and other behavioral variables. Facial expressions can replace sequences of words ("she was dressed [wrinkle nose, stick out tongue]") as well as accompany them [16], and they can serve to help disambiguate what is being said when the acoustic signal is degraded. They do not occur randomly but rather are synchronized to one's own speech, or to the speech of others [11], [22].

Eye gaze is also an important feature of non-verbal communicative behaviors. Its main functions are to help regulate the flow of conversation, signal the search for feedback during an interaction (gazing at the other person to see how she follows), look for information, express emotion (looking downward in case of sadness), or influence another person's behavior (staring at a person to show power)[3], [13].

People also produce hand gestures spontaneously while they speak, and such gestures support and expand on information conveyed by words. The fact that gestures occur at the same time as speech, and that they carry the same meaning as speech, suggests that the production of the two are intimately linked. In fact, not only are the meaning of words and of gestures intimately linked in a discourse, but so are their functions in accomplishing conversational work: it has been shown that certain kinds of gestures produced during conversation act to structure the contributions of the two participants (to signal when an utterance continues the same topic or strikes out in a new direction), and to signal the contribution of particular utterances to the current discourse. It is clear that, like facial expression, gesture is not a kinesic performance independent of speech, or simply a 'translation' of speech. Rather, gesture and speech are so intimately connected that one cannot say which one is dependent on the other. Both can be claimed to arise from a single internal encoding process ([6], [7], [23], [29]).

2.0.1 Example

In this section of the paper we present a fragment of dialogue (the complete dialogue is given in the appendix, and shown on the accompanying videotape), showing rule-generated intonation, gesture, head and eye movements, and their inter-synchronization. This example will serve to demonstrate the phenomena described here, and in subsequent sections we will return to each phenomenon, to explain how rule-generation and synchronization are carried out.

Imagine that Gilbert is a bank teller, and George has asked Gilbert for help in obtaining $50 (as the dialogue is generated automatically the two agents have to specify in advance each of the goals they are working towards and steps they are following (see section 4.1); this explains the redundancy of the dialogue).

Gilbert:	May I ask you something?
George:	Go ahead.
Gilbert:	Do you have an account at this bank?
George:	Yes.
Gilbert:	May I ask you something?
George:	Go ahead.
Gilbert:	Do you have fifty dollars in your account?
George:	Yes.
Gilbert:	May I ask you something?
George:	Go ahead.
Gilbert:	Do you have your checkbook with you?

When Gilbert asks a question, his voice rises. When George replies to a question, his voice falls. When Gilbert asks George whether he has an account at the bank, he stresses the word "account". When he asks whether George has his checkbook with him, he stresses the word "checkbook".

Every time Gilbert replies affirmatively ("yes"), or turns the floor over to Gilbert ("go ahead"), he nods his head, and raises his eyebrows. George and Gilbert look at each other when Gilbert asks a question, but at the end of each question, Gilbert looks up slightly. During the brief pause at the end of affirmative statements the speaker (always George, in this fragment) blinks. To mark the end of the questions, Gilbert raises his eyebrows.

In saying the word 'account", Gilbert forms a kind of box in front of him with his hands: a metaphorical representation of a bank account in which one keeps money. When he says the phrase "this bank," Gilbert points downward, towards the bank floor, with his right hand, while maintaining the 'account' position with his left hand. In saying "checkbook", Gilbert sketches the outlines of a checkbook in the air between him and his listener.

2.1 Functional Significance of Facial Expressions

Movements of the head and facial expressions can be characterized by their placement with respect to the linguistic utterance and their significance in transmitting information [37]. The set of facial movement clusters contains:

- syntactic functions: accompany the flow of speech and are synchronized at the verbal level. Facial movements (such as raising the eyebrows, nodding the head or blinking while saying "do you have your CHECKbook with you") can appear on an accented syllable or a pause.
- semantic functions: can emphasize what is being said, substitute for a word or refer to an emotion (like wrinkling the nose while talking about something disgusting or smiling while remembering a happy event: "it was such a NICE DAY.").
- dialogic functions: regulate the flow of speech and depend on the relationship between two people (smooth turns[1] are often co-occurrent with mutual gaze; e.g at the end of "do you have your checkbook with you?", both interactants look at each other).

These three functions are modulated by various parameters:

- speaker and listener characteristic functions: convey information on speaker's social identity, emotion, attitude, age (friends spend more time looking at each other while talking than a lying speaker who will avoid the other's gaze).

[1] the listener does not interrupt or overlap the speaker

- listener functions: correspond to the listener's reactions to the speaker's speech; they can be signals of agreement, of attention, of comprehension (like saying "I see", "mhmm").

2.2 Communicative Importance of Hand Gestures

Gesture too can be described in terms of its intrinsic relationship to speech. Three aspects of this relationship are described before we go on to speak about the synchronization of the two communicative channels.

First of all, four basic types of gestures occur only during speech ([29] estimates that 90% of all gestures occur when the speaker is actually uttering something).

- Iconics represent some feature of the accompanying speech, such as sketching a small rectangular space with one's two hands while saying "did you bring your CHECKBOOK?"
- Metaphorics represent an abstract feature concurrently spoken about, such as forming a jaw-like shape with one hand, and pulling it towards one's body while saying "you must WITHDRAW money".
- Deictics indicate a point in space. They accompany reference to persons, places and other spatializeable discourse entities. An example is pointing to the ground while saying "do you have an account at THIS bank?".
- Beats are small formless waves of the hand that occur with heavily emphasized words, occasions of turning over the floor to another speaker, and other kinds of special linguistic work. An example is waving one's left hand briefly up and down along with the stressed words in the phrase "go AHEAD".

In some discourse contexts about three-quarters of all clauses are accompanied by gestures of one kind or another; of these, about 40% are iconic, 40% are beats, and the remaining 10% are divided between deictic and metaphoric gestures [29]. And surprisingly, although the proportion of different gestures may change, all of these types of gestures, and spontaneous gesturing in general, are found in discourses by speakers of most languages.

Secondly, there is also a semantic and pragmatic relationship between the two media. Gesture and speech do not always manifest the same information about an idea, but what they convey is always complementary. That is, gesture may depict the way in which an action was carried out when this aspect of meaning is not depicted in speech. For example, one speaker, describing how one deposits checks into a bank account, said "you list the checks" while she depicted with her hands that the deposit slip is to be turned over and turned vertically in order for the checks to be listed in the spaces provided on the back of the slip.

Finally the importance of the interdependence of speech and gesture is shown by the fact that speakers rely on information conveyed in gesture – sometimes even to the exclusion of information conveyed by accompanying speech – as they try to comprehend a story [8].

Nonetheless, hand gestures and gaze behavior have been virtually absent from attempts to animate autonomous or semi-autonomous agents in communicative contexts.

2.3 Synchronization of Gesture, Facial Movements, and Speech

Facial expression, eye gaze and hand gestures do not do their communicative work only within single utterances, but also have inter-speaker effects. The presence or absence of confirmatory feedback by one conversational participant, via gaze or head movement, for example, affects the behavior of the other. A conversation consists of the exchange of meaningful utterances and of behavior. One person punctuates and reinforces her speech by head nods, smiles, and hand gestures; the other person can smile back, vocalize, or shift gaze to show participation in the conversation.

Our contribution to the animation of conversational interaction concentrates on the synchronization of the various verbal and non-verbal behaviors. There is a strong temporal relationship between the channels.

The body and the face do not move at random but in a very coordinated manner. Synchrony implies that changes occurring in speech and in body movements should appear at the same time. For example, when a word begins to be articulated, eye blink, hand movement, head turning, brow raising can occur and can finish at the end of the word.

Synchrony occurs at all levels of speech: phonemic segment, word, phrase or long utterance. Different facial motions are isomorphic to these groups [11], [22]. Some of them are more adapted to the phoneme level, like an eye blink, while others act at the word level, like a frown. In the example "Do you have a checkbook with you?", a raising eyebrow starts and ends on the accented syllables "check", while a blink starts and ends on the pause marking the end of the utterance. Facial expression of emphasis can match the emphasized segment, showing synchronization at this level (a sequence of head nods can punctuate the emphasis). Moreover, some movements reflect encoding-decoding difficulties and therefore coincide with hesitations and pauses inside clauses [12]. Many hesitation pauses are produced at the beginning of speech and correlate with avoidance of gaze (the head of the speaker turns away from the listener) as if to help the speaker to concentrate on what she is going to say.

Gestures occur in synchrony with their semantically parallel linguistic units, although in cases of hesitations, pauses or syntactically complex speech, it is the gesture which appears first ([29]). At the most local level, individual gestures and words are synchronized in time so that the 'stroke' (most energetic part of the gesture) occurs either with or just before the phonologically most prominent syllable of the accompanying speech segment ([23], [29]). At the most global level, we find that the hands of the speaker come to rest at the end of a speaking turn, before the next speaker begins her turn. At the intermediate level, the phenomenon of co-articulation of gestural units is found, whereby gestures are performed rapidly, or their production is stretched out over time, so as to synchronize with preceding and following gestures, and the speech these gestures accompany. An example of gestural co-articulation is the relationship between the two gestures in the phrase "do you have an ACCOUNT at this BANK?": during the word "account", the two hands sketch a kind of box in front of the speaker; however, rather than carrying this gesture all the way to completion (either both hands coming to rest at the end of this gesture, or maintaining the location of the hands in space), one hand remains in the 'account' location while the other cuts short the 'account' gesture to point at the ground while saying 'bank'. Thus, the occurrence of the word "bank", with its accompanying gesture, affected the occurrence of the gesture that accompanied "account".

In the next section of this paper, we outline the work that has been carried out on animation of the face and hands, and synchronization of such systems with speech.

3 Computer animation of conversational behaviors

3.1 Literature on Facial Control Systems

Various systems have proposed different solutions to integrate the different facial expression functions. Most of the systems use **FACS** (Facial Action Coding System) as a notational system [18]. This system is based on anatomical studies. It describes any visible facial movements. An action unit **AU**, basic element of this system, describes the action produced by one or a group of related muscles.

The multi-layer approach [20] has allowed independent control at each level of the system. At the lowest level (geometric level), geometry of the face can be modified using free form deformation techniques. At the highest level, facial animation can be computed from an input utterance. At this level a high-level instruction language is providing allowing the user to specify actions at a higher degree of abstraction (smile while saying "hello") and to perform the corresponding animation.

In M. Patel's model [30] facial animation can also be done at different levels of representation. It can be done either at the muscle level, at the **AU** level or at the script level. For each **AU** the user can select starting and ending points of action, the intensity of action, the start and end tensions and the interpolation method to compute the in-between frames.

Starting from a functional group (lip shapes, conversational signal, punctuator, regulator or manipulator), [31] offer algorithms which incorporate synchrony, create coarticulation effects, emotional signals, and eye and head movements. The authors generate automatically the facial actions corresponding to an input utterance. A conversational signal (movements occurring on accents, like raising of eyebrow) starts and ends with the accented word; while punctuator signal (movement occurring on pause, like smiling) happens on the pause. When a blink is one of these signals it is synchronized at the phoneme level. Head nods and shakes appear on accent and pause. The head of the speaker turns away from the listener at the beginning of a speaking turn and turns toward to the listener at end of a speaking turn to signal a change of turn. Other signals such as emblems, and emotional emblems are performed consciously and must be specified by the user.

Building a user-interface, [39] propose a categorization of facial expressions depending on their communicative meaning. For each of the facial functions a list of facial displays is done (for example, question mark corresponds to raising eyebrows, remembering corresponds to eyebrows action, eye closure and one side of mouth pull back). A user talks to the 3D synthetic actor. A speech system recognizes the words and generates an answer with the appropriate facial displays. Grammar rules, a small vocabulary set and a specific knowledge domain are part of the speech analysis system. The responses by the 3D actor are selected from a pre-established set of utterances. The appropriate facial displays accompanying the answer follow the analysis of the conventional situation (e.g. if the user's speech is not recognized the 3D actor will answer with a "not-confident" facial display).

3.2 Literature on Gesture Animation

The computer graphics literature is rather sparse on the topic of gesture animation. Animators frequently use key parameter techniques to create arm and hand motions. Rijpkema and Girard [35] created handshapes automatically based on the object being gripped. The Thalmanns [19, 28] improved on the hand model to include much better skin models and deformations of the finger tips and the gripped object. Lee and Kunii [24] built a system that includes handshapes and simple pre-stored facial expressions for American Sign Language (ASL) synthesis.

Dynamics of arm gestures in ASL have been studied by Loomis et al [27]. Chen et al [9] constructed a virtual human that can shake hands with an interactive participant. Lee et al [25] automatically generate lifting gestures by considering strength and comfort measures. Moravec and Calvert [4] constructed a system that portrays the gestural interaction between two agents as they pass and greet one another. Behavioral parameters were set by personality attribute "sliders" though the interaction sequence was itself pre-determined and limited to just one type of non-verbal encounter.

No literature appears to exist on the integration of facial animation with gesture generation in the context of intonationally correct speech.

4 Overview of system

In the current system, a model of face-to-face interaction is used to generate all of the behaviors implemented, from the informational status of intonation to the communicative function of head nods, gaze, and hand gestures. Additionally, however, this system implements two agents whose verbal and nonverbal behaviors are integrated not only within turns, but across speakers.

In the remaining parts of the paper we explain the different boxes of figure 1. We start from the top of the figure and work towards its bottom.

4.1 Dialogue Planner

The text of this dialogue may be automatically generated on the basis of a database of facts describing the way the world works, a list of the goals of the two agents, and the set of beliefs of those two agents about the world, including the beliefs of the agents about one another [32]. In this instance the two agents have different beliefs about the world (George believes that he has $50 in his account, and Gilbert does not), and goals that change over the course of the dialogue (Gilbert comes to have the goal of helping George get $50).

Text is generated and pitch accents and phrasal melodies are placed on generated text roughly as outlined in [38] and [33]. This text is converted automatically to a form suitable for input to the AT&T Bell Laboratories TTS synthesizer ([26]). When the dialogue is generated, the following information is saved automatically:

- the timing of the phonemes and pauses
- the type and place of the accents
- the type and place of the gesture

This speech and timing information will be critical for synchronizing the facial and gestural animation.

4.2 Symbolic Gesture Specification

Utterances are annotated according to how their semantic content could relate to a spatial expression (literally, metaphorically, spatializeably, or not at all). Further, references to entities are classified according to discourse status as either new to discourse and hearer (indefinites), new to discourse but not to hearer (definites on first mention), or old (all others) [34]. According to the following rules, these annotations, together with the earlier ones, determine which concepts will have an associated gesture. Gestures that represent something (iconics and metaphorics)

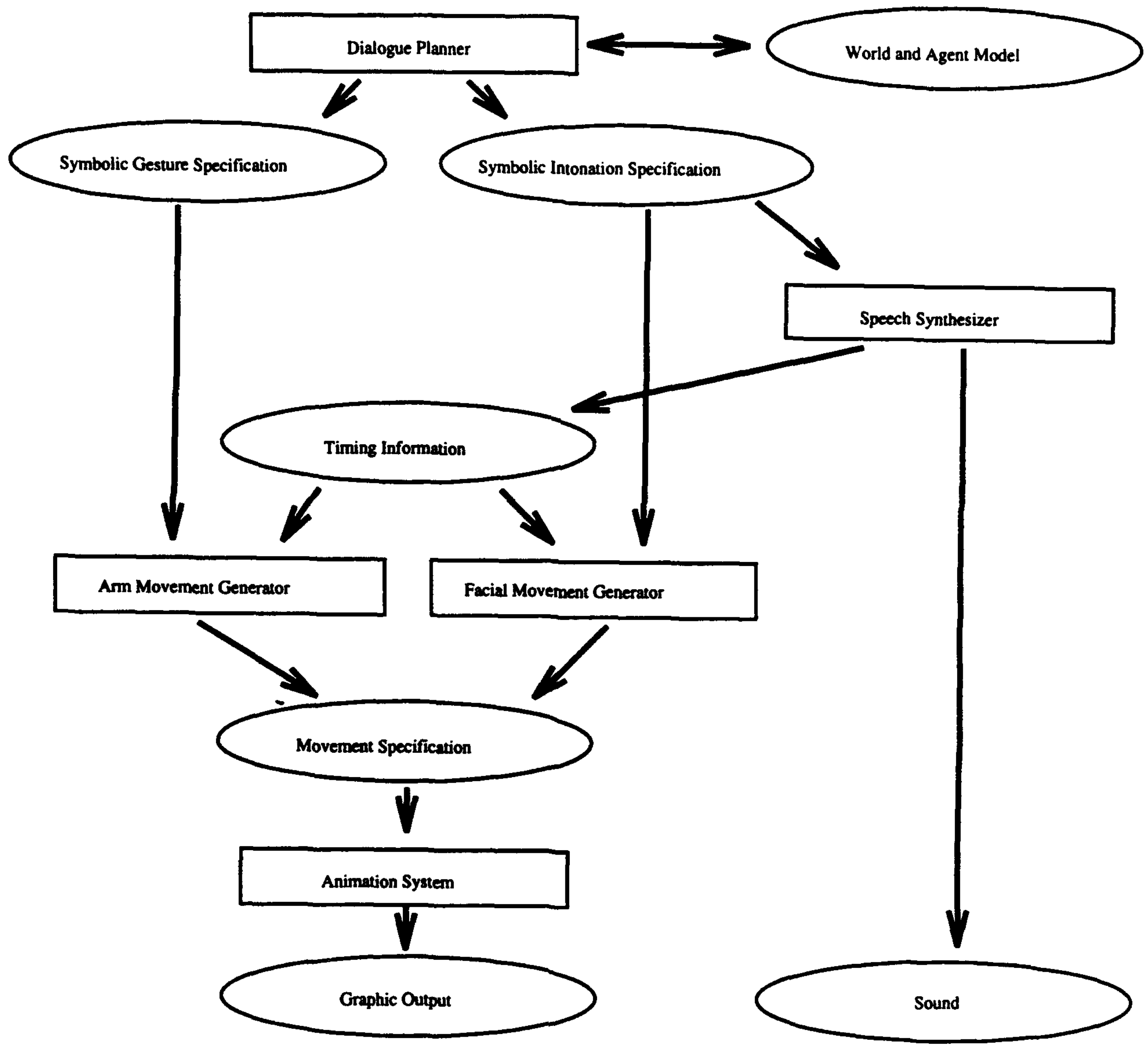


Figure 1: interaction of components

********Slide 1: 4 frames representing 4 different gestures *********

Figure 2: Examples of symbolic gesture specification

are generated for rhematic verbal elements (roughly, information not yet spoken about) and for hearer new references, provided that the semantic content is of an appropriate class to receive such a gesture: words with literally spatial (or concrete) content get iconics (e.g. "checkbook"), those with metaphorically spatial (or abstract) content get metaphorics (e.g. "plan"); words with physically spatializeable content get deictics (e.g. "this"). Meanwhile, beat gestures are generated for such items when the semantic content cannot be represented spatially, and are also produced accompanying discourse new definite references (e.g. "fifty-SEVEN dollars"). If a representational gesture is called for, the system accesses a dictionary of gestures (motion prototypes) for concepts in order to determine the symbolic representation of the particular gesture to be performed.

In figure 2, we see examples of how symbolic gestures are generated from discourse content.

1. "Shall we make a PLAN?"

   - In the first frame, a metaphoric gesture (the common *conduit* gesture, representing the plan as an entity that can be presented to the listener) is generated because of the first mention (new to hearer) of the abstract notion 'make a plan'.

2. I suggest that you WRITE a check

   - In the second frame, an iconic gesture (representing writing on a piece of paper) is generated from the first mention of the concrete action of 'writing a check'.

3. Your account contains THREE dollars

   - In the third frame, an iconic gesture (the *emblematic* gesture understood to mean 'three') is generated from the first mention (new to hearer) of the entity 'three dollars'.

4. three dollars is LESS than fifty dollars

   - In the fourth frame, a beat gesture (a movement of the hand up and down) is generated from the first mention of the notion 'less than', which cannot be represented spatially.

After this gestural annotation of all gesture types, and lexicon look-up of appropriate forms for representational gestures, information about the duration of intonational phrases (acquired in speech generation) is used to time gestures. First, all the gestures in each intonational phrase are collected. Because of the relationship between accenting and gesturing, in this dialogue at most one representational gesture occurs in each intonational phrase. If there is a representational gesture, its preparation is set to begin at or before the beginning of the intonational phrase, and to finish at or before the next gesture in the intonational phrase or the nuclear stress of the phrase, whichever comes first. The stroke phase is then set [illegible] ncide with the nuclear stress of the phrase. Finally, the relaxation is set to begin no sc [illegible] an the end of the stroke or the end of the last beat in the intonational phrase, with the [illegible] of relaxation to occur around

**************Slide 2: 4 frames representing the two agents in close up.**********

Figure 3: Facial expressions corresponding to the utterance 'go ahead'.

the end of the intonational phrase. Beats, in contrast, are simply timed to coincide with the stressed syllable of the word that realizes the associated concept. When these timing rules have applied to each of the intonational phrases in the utterance, the output is a series of symbolic gesture types and the times at which they should be performed. These instructions are used to generate motion files that run the animation system.

4.3 Symbolic Intonation Specification

P. Ekman and his colleagues characterize the set of semantic and syntactic facial expressions depending on their meaning [15]. Many facial functions exist (such as manipulators that correspond to biological need of the face (wetting the lips); emblems and emotional emblems that are facial expressions replacing a word, an emotion) but only some are directly linked to the intonation of the voice. A conversational signal corresponds to the movement appearing on an accent; a punctuator signal appears on a pause; We will illustrate the conversational signal function by an example. Let us consider the sentence "Go ahEAd". The segment 'ea' receives the pitch accent. The figure 3 illustrates this example. The first frame is the pronunciation of the segment 'go'; the mouth is puckered to form the sound 'o'. The second frame corresponds to the first phonemic segment 'a' of the word 'ahead'. Since the pitch accent is on the second syllable, a conversational signal occurs on the word. The raising of eyebrows is a common form of this signal as is a blink. The eyebrows start raising on the first syllable of the accented word. The pitch of the movement corresponds to the accented vowel 'ae' and the eyes are closed for a blink (frame 3). The last frame shows the end of the pronunciation of the word 'ahead'. The eyebrows are back to their 'ne[illegible]' position and the eyes are open.

4.4 Arm Movement [illegible]erator

The gesture command syntax sends information about the timing, shape, and position of the hands and arms to the animation system. The animation process produces a file of motions to be carried out by this figure. Starting with the given gestures and their timing, constraints imposed by the human agent's abilities and the speech rate modify the motion sequence for a proper co-articulation effect.

The first step is the reading of data from a sequence of phonemes, intonational information, and gestures.

This data is output in the following form.

88	1	0	5455	5455	st(checkbook)
d	0	7	5455	5462	
uu	2	9	5462	5471	
88-89	0	0	5471	5471	
y	0	6	5471	5477	
uu	2	9	5477	5486	
89-90	0	0	5486	5486	
h	0	10	5486	5496	
ae	1	15	5496	5511	
v	0	8	5511	5519	
90-91	1	0	5519	5519	
y	0	6	5519	5525	
uu	2	9	5525	5534	
rr	0	6	5534	5540	
	0	75	5679	5754	

In this example, the phonemes on the far left compose the phrase "do you have your" of the utterance "Do you have your checkbook with you?" The line which concludes "st(checkbook)" indicates which gesture will be produced during the production of the sentence; in the continuation of this utterance (not shown) the symbols "pr", "sk", and "rx" refer to preparation, stroke, and relaxation phases of a gesture respectively. Where two gestures occur during a single sentence, gestures are marked following the stroke or relaxation coding in the file.

In this instance, the presence of a gesture is signalled simultaneously with the preparation of that gesture; furthermore, it follows a preceding gesture which presumably has been executed before the signalling of a further gesture. If no gestures are signalled at the beginning of a utterance, we can assume that none will occur. The remaining columns denote timing information for each phoneme and other speech information such as accents. The fourth and fifth columns refer to the time that each phoneme starts and ends, respectively. These are of particular interest since these times will parameterize gesture.

Due to the structure of the conversation, where the speakers alternate turns, we assume similar alternation in gesturing. (Gesturing by listeners is almost non-existent [29].) For the purposes of gesture generation, phoneme information (the first three columns of the example above) can be eliminated; however, utterance barriers, which are denoted by intonational information in the sixth column, must be interpreted both to provide an envelope for the timing of a particular gesture or sequence of gestures and to determine which speaker is gesturing.

Once timing information has been pulled out in this manner, gestures selected by the dialogue generator can be checked against canonical times for these gestures. In the lexicon, timing information has been recorded for both the length of preparation time required for a gesture, and the total execution time of a gesture. If the time available for a gesture is significant enough to encompass these times, then a production of a complete gesture at the indicated time may occur. For example, the gesture which accompanies the statement

"Do you have your [checkbook] with you?"

has sufficient time to execute: it is the only gesture occurring in the phrase, as is evidenced by the timing information given in files produced by the first section. However, if this timing is insufficient to allow for full gesture production, then the gesture must be foreshortened to allow for the reduced available timing.

The most common reason for foreshortening is anticipation of the next gesture to be produced in a discourse. In anticipatory co-articulation effects, most often the relaxation phase of the foreshortened gesture and preparation phase of the next gesture become one. This process can be seen in the gestures accompanying the phrase

"Do you have [an account] at [this bank]?"

"[An account]" is produced .90 seconds into the phrase, and "[this bank]" is generated at 1.9 seconds. This causes some foreshortening in the relaxation process during the first gesture, from which the second gesture is then produced. The same effect can be seen in the phrase

"I suggest that you [write a check] and [withdraw] fifty dollars from your account."

Co-articulation constraints – synchronizing the gestures with intonational phrases and surrounding gestures – may actually cause the given gestures to be aborted if too little time is available for production given the physical constraints of the human model.

Once a sequence of gestures has been selected and timing information generated, this information must be put into a motion file (or series of motion files, in the case of a full dialogue) which will be read by the animation software. The motion file is composed of several separate units which take parameters such as depth, height, breadth, position, handshape, and so forth to create gestures at any position in space. One part of the motion file generated by parsing the sentence "Do you have [an account] at [this bank]?" follows. The lines specifying figure (referring to which figure is gesturing), start-time, and duration are generated by parsing the speech generation file to determine who is speaking and when the arm, wrist, or fingers begin and stop moving.

```
/* Motions, #20 */ [20th utterance]
motion gesture_guy_0 {
figure = (figure)Gilbert;
type = "arm gesture";
starttime = 0.90sec;
duration = 0.73sec;
velocitycontrol = "constant";
weightfunction = "increase";
data = ("right", "near","chest","center right",2.00);
}

motion gesture_guy_2 {
figure = (figure)Gilbert;
type = "wrist gesture";
starttime = 0.90sec;
duration = 0.73sec;
velocitycontrol = "constant";
weightfunction = "decay";
data = ("right", "forward","left",2.00);
}

motion gesture_guy_9 {
figure = (figure)Gilbert;
type = "hand gesture";
starttime = 1.90sec;
duration = 0.73sec;
velocitycontrol = "constant";
data = ("right", "D", 95.00);
}
```

Files such as these accompany every utterance spoken by a figure in the animation, whether or not they contain specifications for gesture. These :notion files are then interpreted by the gesture animation system, described below. In cases where there is no gesture, the motion files specify that the hands remain at rest. The rest position can be changed so that the hands remain closer to the sides of the body, or closer to the chest. (In the video that accompanies this paper, the rest position chosen is not optimal.)

4.5 Gesture Motion Specification

The graphics-level gesture animation system accepts gesture instructions containing information about the location, type, timing, and handshape of individual gestures. Based on the current location of the hands and arms in space, the system will attempt to get as close as possible to the gesture goals in the time allowed, but may mute motions or positionings because it cannot achieve them in time (co-articulation effects). This animation system calls upon a library of predefined handshapes which form the primitives of hand gesture. These handshapes were chosen to reflect the shapes most often found in gesture during conversational interaction ([23]). The animation system also calls upon separate hand, arm and wrist control mechanisms.

The gesture system is divided into three parts: hand shape, wrist control, and arm positioning. The first, hand shape, relies on an extensible library of hand shape primitives for the basic joint positions, but allows varying degrees of relaxation towards a neutral hand position. The speed at which the hand may change shape is also limited to allow the modelling of hand shape co-articulation. Large changes in hand position are restricted as less time is allotted for the hand movement, forcing faster hand gestures to smooth together. Instructions to the hand control system may be seen in the motion file excerpt above.

The wrist control system allows the wrist to maintain and change its position independently of what complex arm motions may be occurring. The wrist is limited within the model to a physically realistic range of motion. Wrist direction is specified in terms of simple directions relative to the gesturer, such as "point the fingers of the left hand forward and up, and the palm right". Instructions to the wrist control system may be seen in the sample motion file.

The arm motion system accepts general specifications of spatial goals and drives the arms towards those goals within the limits imposed by the arm's range of motion. The arm may be positioned by using general directions like "chest-high, slightly forward, and to the far left". Instructions to the arm control system appear in the sample motion file.

The expressiveness of an individual's gesturing can be represented by adjusting the size of the gesture space of the graphical figure. In this way, parameters such as age (children's gestures are larger than adults') and culture (in some cultures gestures tend to be larger) can be implemented in the gesture animation.

4.6 Facial Movement Generator

We use Pelachaud's algorithm [31] for computing the set of facial expressions that correspond to the set of semantic and syntactic functions 4.3. In this model, each facial expression is represented by two parameters: *its time of occurrence* and *its type*. Different speaker personalities can be obtained by varying these two parameters. For example a persuasive p^rson can punctuate each accented word with raising eyebrows, while another person will not. Eye behavior, which is discussed below, is an important characteristic of a personality. For the dialogic functions we

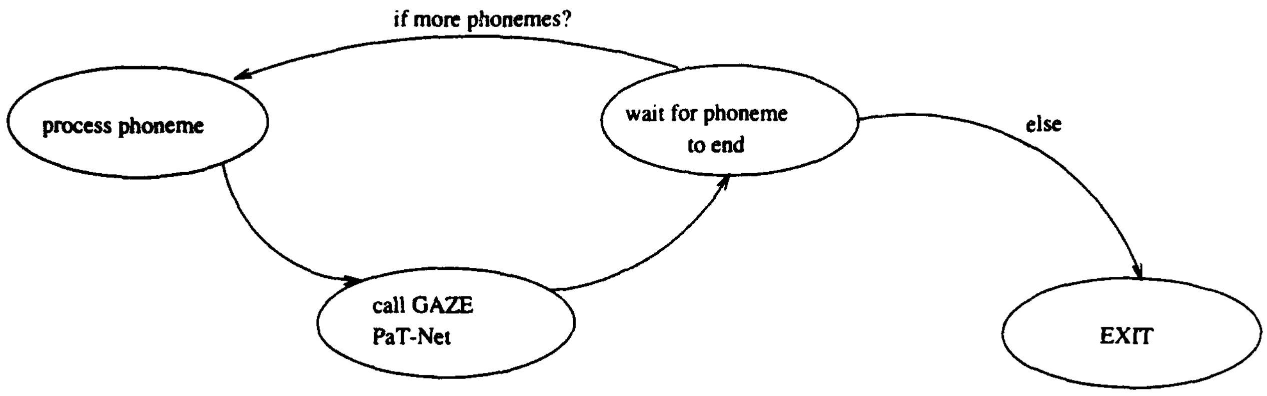


Figure 4: Pat-Net that synchronizes the gaze network with the dialogue at the phoneme level.

have first examined the synchrony of eye and head movement. The next sections present our model and then show how each facial function is implemented.

4.6.1 The Underlying Coordination Model

Facial interaction between agents and synchronization of head and eye movements to the dialogue for each agent are accomplished using Parallel Transition Networks (PaT-Nets), which allow facial coordination rules to be encoded as simultaneously executing finite state automata. PaT-Nets can call for action in the simulation and make state transitions either conditionally or probabilistically. Pat-Nets are scheduled into the simulation with an operating system that allows them to invoke or kill other PaT-Nets, sleep until a desired time or until a desired condition is met, and synchronize with other running nets by waiting for them to finish or by waiting on a shared semaphore.

In addition, the PaT-Net notation is object oriented with each net defined by a *class* with actions and transition conditions as ***methods***. The running networks are instances of the PaT-Net class and can take parameters on instantiation. This notation allows Pat-Nets to be hierarchically organized and allows constructing new nets by combining existing nets or making simple modifications to existing nets.

As an example, consider the PaT-Net in Figure 4, which synchronizes an agent's head and eye movements with the dialog at the phoneme level. The example net defines the `ProcessPhoneme` class and an instance of this class is created for each agent (one for the listener and one for the speaker) with the agent's name as a parameter. In the `wait for phoneme` node, each net instance blocks until the current phoneme ends (or until the simulation starts on startup). If there are more phonemes the net makes a transition to the `process-phoneme` node where it sets appropriate probabilities for the net's agent given the agent's current role. If there are no more phoneme the net exits. From the `process-phoneme` node the net always makes a transition to the `call GAZE` node, where it calls the head and eye movement PaT-Net (figure 5) and waits for it to exit before continuing (it effectively calls the gaze PaT-Net as a subroutine since it waits for it to finish, but PaT-Nets can run in parallel, however, when appropriate). When the `GAZE` PaT-Net exits, the phoneme PaT-Net returns to the `wait-for-phoneme` node, where it blocks until the speaker's current phoneme ends in the simulation.

Eye and head behaviors are implemented as specified in the following section 4.6.2 with all face and eye movement behavior for an individual encoded in a single PaT-Net. A PaT-Net instance is created to control each agent with appropriate parameters. Then as agents' PaT-Nets synchronize the agents with the dialogue and interact with the unfolding simulation they schedule activity that achieves a complex observed interaction behavior.

4.6.2 Dialogic Functions

Eye movements can be classified into four primary categories depending on their role in the conversation [1], [17], [10]. Eye movements:

1. **feedback:** are used to collect and seek feedback;
2. **planning:** correspond to the first phase of a turn when the speaker organizes her thoughts;
3. **comment:** accompany and comment speech, by occuring in parallel with accent and emphasis;
4. **control:** control the communication channel and function as a synchronization signal: responses may be demanded or suppressed by looking at the listener.

Each of these appears as a sub-network in the PaT-Net. Their behavior will be discussed below. Figure 5 outlines the high-level PaT-Net for eye and head movement control for a single agent. It contains the four dialogic functions, their nodes that define each function, and their associated actions. We shall follow this figure in the forthcoming explanations.

4.6.3 Example

As an example, consider the utterance:

```
I disagrEE <pause>
your account contains thrEE dollars.
```

where capital letters (EE) correspond to accented segments.

For each phonemic segment, the PaT-Net in Figure 5 is executed for each agent (speaker and listener, here respectively, Gilbert and George). At each node (state) the specified action is performed and a transition is made to a new state either based on a known probability for a pattern of activity (see Section 4.7) or on properties of the simulation. Probabilities appropriate for each agent given the current role as listener or speaker are set for the PaT-Net before it executes. At each turn change, the probabilities change values accordingly. The actions performed during an utterance are described in detail below. Figure 6 shows different frames of the animation. The top left corner represents the beginning of the turn: Gilbert says "I". The next frame is on the accented word "disagree": Gilbert and George look at each other and Gilbert's eyebrows start to raise to mark the accent. The third frame corresponds to the accented phonemic segment "ee" of disagree. Different actions occur on the pitch segment: a blink, a raising eyebrow and a head nod. The leftmost frame on the middle row represents the 'within-turn signal': Gilbert has the turn but is pausing; he looks at George. The next frame shows the 'speaker-continuation-signal': Gilbert keeps the turn and turns his head away from Gilbert as he starts pronouncing the second intonation phrase "your account". The last frame on this row corresponds to "contains". On the bottom left frame, Gilbert is saying "three" which receives a pitch accent: as previously, Gilbert blinks and raises his eyebrows, but he does not nod since the random value corresponding to the node **accent** of the sub-network **comment** does not exceed the given probability threshold. At the end of the turn Gilbert looks at George while George looks up as shown in the next frame. The last frame is the beginning of the next turn.

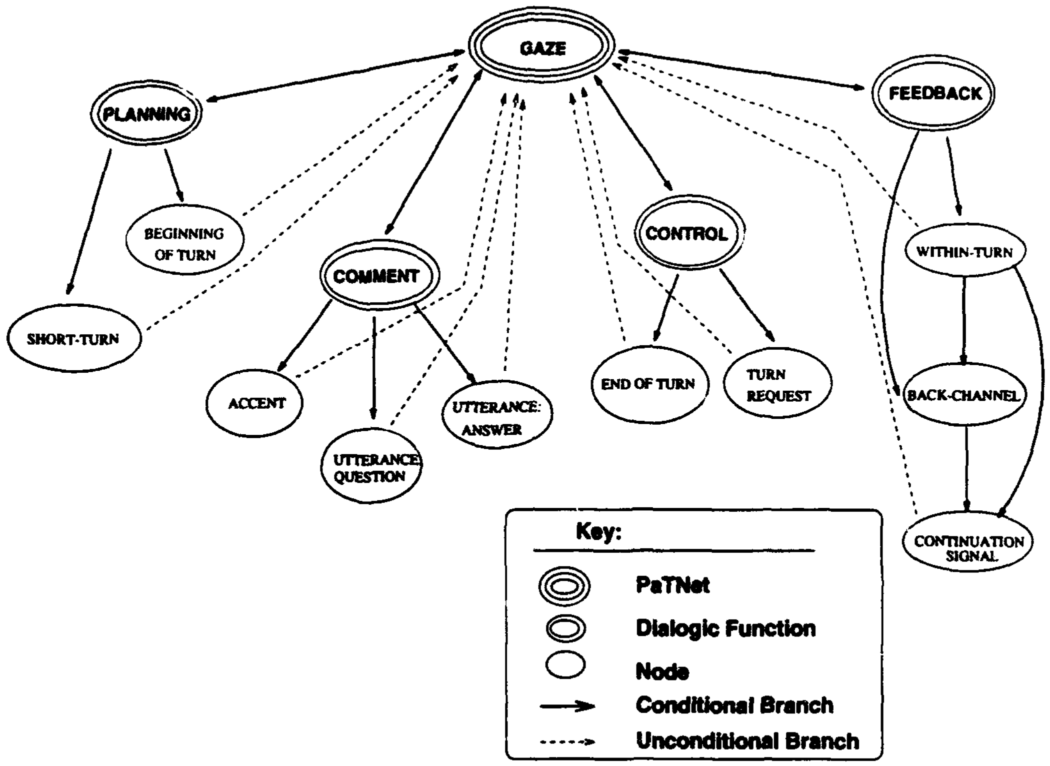


Figure 5: The eye/head movement PaT-Net: actions are defined in the nodes; conditional and probabilistic transitions occur on arcs.

********Slide 3: 9 frames representing the two agents in close up.*********

Figure 6: Facial expressions and gaze behavior corresponding to: "I disagree <pause> your account contains three dollars".

4.6.4 Planning Phase

When a person speaks, two phases in the speech can be defined: the planning phase and the execution phase [5]. During the planning phase the speaker looks for and defines what she is going to say. Throughout this phase speech is more hesitant due to concurrent cognitive processes [12]. The speaker has a tendency to look away in order to prevent an overload of information. This phase corresponds to the Speaker-State-Signal as defined in the turn-taking system. On the other hand, during the execution phase, the speaker knows what she is going to say and looks more at the listener. For a short turn (duration less than 1.5 sec.), the speaker and the listener establish eye contact [1].

For the first few phonemes of the beginning of the example utterance [2](in our example it corresponds to "I disagr"), the sub-network **planning** is applied. This utterance is not short so the node **`short-turn`** is not entered. But the node **`beginning-turn`** is entered; the condition of being in a beginning of turn is true but its probability did not allow the action **`speaker looks away`** to be applied. Therefore the speaker (Gilbert) keeps his current gaze direction (looking at George). On the other hand when Gilbert starts saying:

`I suggest that you write a check` *<pause>*
`and withdraw fifty dollars from your account.`

the node **`beginning-turn`** is entered and executed since the condition is true and the probability allows it.

4.6.5 Comment

Three functions characterize this stage. Accented or emphasized items are punctuated by facial expressions (mainly eyebrow raising) and head nods. Glances by the speaker toward the listener act to emphasize particular words of a phrase. The speaker gazes at the listener more also when she asks a question. She looks up at the end of the question. During a pause by the speaker, the listener looks up. In our example, on accented items ("disagrEE" and "thrEE"), the node **`accent`** of the sub-network **comment** is reached; the actions **`speaker looks at the listener`** and **`head nod`** are performed by Gilbert. As before, the instantiation of an action depends on its probability. The system easily represents the parallel agent actions.

4.6.6 Control

Dialogic regulators correspond to how people take speaking turns in a conversation or ritual meeting. Much study has been given to turn-taking systems [13]. In the control sub-network we consider two types of signal in the turn-taking system:

1. **End-of-Turn** (Speaker-Turn-Signal): the speaker wants to give her turn of speaking to the listener. The speaker turns her head toward the listener at the end of the utterance.
2. **Listener-Turn-Request**: the listener asks for the turn. She looks up at the speaker, shows impatience, and so on.

In our example at the end of the utterance[3] (corresponding to "dollars" here) the sub-network **control** is entered. Two actions are considered. The node **`end of turn`** corresponds to action performed by the speaker: **`speaker looks at listener`**. The other node **`turn request`** affects the listener; the action **`listener looks at the speaker and up`** is performed.

[2] a beginning of a turn is defined as all the phonemes between the first one and the first accented segment.
[3] end of turn is defined as all the phonemes between the last accented segment and the last phonemes.

4.6.7 Feedback

This sub-network involves the speaker and the listener. Speakers look up at grammatical pauses to obtain feedback on how utterances are being received. The listener can emit different reaction signals to the speaker's speech [14], [21].

A sequence of actions characterizes this sub-network:

- **Speaker-Within-Turn:** is used when the speaker wants to keep her speaking turn, and reassure herself that the listener is following. It occurs at the completion of a grammatical clause; the speaker turns her head toward the listener. It is frequently followed by a listener **Backchannel** (see below) which in turn may be followed by a Speaker-Continuation-Signal if the speaker wants to keep her turn. This sequence of signals corresponds to the feedback function.

- **Backchannel:** is emitted by the listener after a Speaker-Within-Turn. It can consist of the listener looking at the speaker, smiling, nodding, vocalizing, and so on.

- **Speaker-Continuation-Signal:** frequently follows a Speaker-Within-Turn. In such a case, the speaker turns her head away from the listener.

If the speaker doesn't emit a within-turn-signal, the listener can still emit a backchannel which in turn may be followed by a speaker-continuation signal. But the probability of action of the listener (emitting a backchannel) varies with the action of the listener [13]; in particular, it decreases if no signal has occurred from the speaker. In this way the listener reacts to the behavior of the speaker.

The two intonational phrases of our example (*I agree* and *your account contains 3 dollars* are separated by a pause; this corresponds to a within-turn situation. The sub-network **feedback** is entered. If the probability allows it, the action **`speaker looks at the listener`** is performed[4]). After a delay (0.2 sec., as specified by the program), the node **`backchannel`** is reached. Once more the program checks the probabilities associated with the actions. Two actions can happen: **`listener looks at the speaker`** and/or **`the listener nods`**. In either case, the final step within the **feedback** sub-network is reached after some delay. The action **`speaker looks away from the listener`** is then performed.

4.7 Facial Motion Specification

We discuss now the instantiation of the fa l movements generated in a previous stage (as discussed in 4.6). As discussed earlier, the GAZE PaT-Net in Figure 5 is run for each agent at the beginning of every phoneme. Depending on the course taken through the GAZE network due to probabilistic branching and environmental state, the net may commit its agent to a variety of actions such as a head nod or a change in the gaze point. A change in the gaze is accomplished by supplying the human model with a 3D coordinate at which to look and a time in which to move – the scheduled motion then begins at the current point in the simulation and has the specified duration. A head nod is accomplished by scheduling a sequence of joint motions for the neck, supplying both the angle and the angular velocity for each nod cycle. Note that the gaze controller schedules motions as they are necessary by reacting to the unfolding simulation (in fact, it does this in semi-real time) and does not have to generate all motions in advance. This makes the gaze controller easy to extend and easy to integrate with the rest of the system.

[4] In the case the action is not performed, the arc going to the node backchannel is immediately traversed without waiting for the next phonemic segment

Different functions may be served by the same action, which differ only in their timing and amplitude. For example, when punctuating an accent, the speaker's head nod will be of larger amplitude than the feedback head nods emitted by the listener. Different head nod functions may also be characterized by varying numbers of up/down cycles. For eye movement, the gaze direction is sustained by calling for the agent to look at a pre-defined point in the environment until a change is made by another action.

A person can have the floor talking or pausing, but loses it as soon as the other person starts talking. There are 3 possible states per person while having the floor. If Speaker has the floor: Speaker talks and Listener pauses, both of them are talking and both of them are pausing. For each of these states, Speaker and Listener can gaze at each other or not. This gives us 12 possibilities, or 24 per dyad. We can then compute the probability of being in each of these states.

Most of the nodes of the Pat-Net can be characterized by a certain set of states. For example the occurrence of a "within-turn signal" as we defined it corresponds to the action: person1 looks at the person2 while having the floor and pausing. These state sets correspond to a sub-matrix. We compute the probability of such sub-matrix in relation to the particular state (having the floor and pausing) to arrive at a probability of occurrence. We do such a computation for all the other nodes of the Pat-Net. This information is used to determine the rules and transitional probabilities for actions in Pat-Nets.

5 Conclusions

We have described and implemented a system to *automatically* animate conversations between multiple human-like agents. The conversation is created by a dialogue planner that produces the text as well as the intonation of the utterances. The speaker/listener relationship, the text, and the intonation in turn drive facial expressions, lip motions, eye gaze, head motion, and arm gesture generators. The facial motion models are driven and synchronized by parallel transition networks. Coordinated arm, wrist, and hand motions are invoked to create semantically meaningful gestures. This system is an important addition to the field of graphic animation of autonomous agents in that, along with speech, it automatically generates appropriate timings and synchronized motions for face, eyes, head, arms, and hands. Such important features of real conversations are therefore not left to chance nor the intuitions of the animator. Timing, intonation, and accents are known to be crucial components of conversational speech. Automatically generating this information allows an interactive dialogue animation to be created; for a non-real-time animation much guess-work in the construction of appropriate motions can be avoided. The resulting motions can be used as is – as demonstrated in the accomanying video – or the actions and timings can be used as a cognitively and physiologically justified guide to further animator refinement of the conversation and the participants' interactions.

References

[1] M. Argyle and M. Cook. *Gaze and Mutual gaze.* Cambridge University Press, 1976.

[2] Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors. *Making Them Move: Mechanics, Control, and Animation of Articulated Figures.* Morgan-Kaufmann, San Mateo, CA, 1991.

[3] G.W. Beattie. Sequential temporal patterns of speech and gaze in dialogue. In T.A. Sebeok and J. Umiker-Sebeok, editors, *Nonverbal Communication, Interaction, and Gesture*. The Hague, New-York, 1981.

[4] Tom Calvert. Composition of realistic animation sequences for multiple human figures. In Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, pages 35–50. Morgan-Kaufmann, San Mateo, CA, 1991.

[5] J. Cappella. personal communication, 1993.

[6] Justine Cassell. *The Development of Time and Event in Narrative: Evidence from Speech and Gesture*. PhD thesis, University of Chicago, 1991.

[7] Justine Cassell and David McNeill. Gesture and the poetics of prose. *Poetics Today*, 12:375–404, 1992.

[8] Justine Cassell, David McNeill, and Karl-Erik McCullough. Kids, don't try this at home: Experimental mismatches of speech and gesture. presented at the International Communication Association annual meeting, 1993.

[9] D. T. Chen, S. D. Pieper, S. K. Singh, J. M. Rosen, and D. Zeltzer. The virtual sailor: An implementation of interactive human body modeling. In *Proc. 1993 Virtual Reality Annual International Symposium*, Seattle, WA, September 1993. IEEE.

[10] G. Collier. *Emotional expression*. Lawrence Erlbaum Associates, 1985.

[11] W.S. Condon and W.D. Osgton. Speech and body motion synchrony of the speaker-hearer. In D.H. Horton and J.J. Jenkins, editors, *The perception of Language*, pages 150–184. Academic Press, 1971.

[12] A.T. Dittmann. The body movement-speech rhythm relationship as a cue to speech encoding. In Weitz, editor, *Nonverbal Communication*. Oxford University Press, 1974.

[13] S. Duncan. Some signals and rules for taking speaking turns in conversations. In Weitz, editor, *Nonverbal Communication*. Oxford University Press, 1974.

[14] S. Duncan and D.W. Fiske. *Interaction Structure and Strategy*. Cambridge University Press, 1985.

[15] P. Ekman. Movements with precise meanings. *The Journal of Communication*, 26, 1976.

[16] P. Ekman. About brows: emotional and conversational signals. In M. von Cranach, K. Foppa, W. Lepenies, and D. Ploog, editors, *Human ethology: claims and limits of a new disipline: contributions to the Colloquium*, pages 169–248. Cambridge University Press, Cambridge, England; New-York, 1979.

[17] P. Ekman and W. Friesen. *Unmasking the Face: A guide to recognizing emotions from facial clues*. Prentice-Hall, Inc., 1975.

[18] P. Ekman and W. Friesen. *Facial Action Coding System*. Consulting Psychologists Press, Inc., 1978.

[19] Jean-Paul Gourret, Nadia Magnenat-Thalmann, and Daniel Thalmann. Simulation of object and human skin deformations in a grasping task. *Computer Graphics*, 23(3):21–30, 1989.

[20] P. Kalra, A. Mangili, N. Magnenat-Thalmann, and D. Thalmann. Smile: A multilayered facial animation system. In T.L. Kunii, editor, *Modeling in Computer Graphics*. Springer-Verlag, 1991.

[21] A. Kendon. Some functions of gaze-direction in social interaction. *Acta Psychologica*, 26:22–63, 1967.

[22] A. Kendon. Movement coordination in social interaction: some examples described. In Weitz, editor, *Nonverbal Communication*. Oxford University Press, 1974.

[23] Adam Kendon. Gesticulation and speech: Two aspects of the process of utterance. In M.R.Key, editor, *The Relation between Verbal and Nonverbal Communication*, pages 207–227. Mouton, 1980.

[24] Jintae Lee and Tosiyasu L. Kunii. Visual translation: From native language to sign language. In *Workshop on Visual Languages*, Seattle, WA, 1993. IEEE.

[25] Philip Lee, Susanna Wei, Jianmin Zhao, and Norman I. Badler. Strength guided motion. *Computer Graphics*, 24(4):253–262, 1990.

[26] Mark Liberman and A. L. Buchsbaum. Structure and usage of current Bell Labs text to speech programs. Technical Memorandum TM 11225-850731-11, AT&T Bell Laboratories, 1985.

[27] Jeffrey Loomis, Howard Poizner, Ursula Bellugi, Alynn Blakemore, and John Hollerbach. Computer graphic modeling of American Sign Language. *Computer Graphics*, 17(3):105–114, July 1983.

[28] Nadia Magnenat-Thalmann and Daniel Thalmann. Human body deformations using joint-dependent local operators and finite-element theory. In Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, pages 243–262. Morgan-Kaufmann, San Mateo, CA, 1991.

[29] David McNeill. *Hand and Mind: What Gestures Reveal about Thought*. University of Chicago, 1992.

[30] M. Patel. *Making FACES*. PhD thesis, School of Mathematical Sciences, University of Bath, Bath, AVON, UK, 1991.

[31] C. Pelachaud, N.I. Badler, and M. Steedman. Linguistic issues in facial animation. In N. Magnenat-Thalmann and D. Thalmann, editors, *Computer Animation '91*, pages 15–30. Springer-Verlag, 1991.

[32] Richard Power. The organisation of purposeful dialogues. *Linguistics*, 1977.

[33] Scott Prevost and Mark Steedman. Generating contextually appropriate intonation. In *eacl93*, pages 332–340, Utrecht, 1993.

[34] Ellen F. Prince. The ZPG letter: Subjects, definiteness and information status. In S. Thompson and W. Mann, editors, *Discourse description: diverse analyses of a fund raising text*, pages 295–325. John Benjamins B.V., 1992.

[35] Hans Rijpkema and Michael Girard. Computer animation of hands and grasping. *Computer Graphics*, 25(4):339–348, July 1991.

[36] Barbara Robertson. Easy motion. *Computer Graphics World*, 16(12):33–38, December 1993.

[37] Klaus R. Scherer. The functions of nonverbal signs in conversation. In H. Giles R. St. Clair, editor, *The Social and Physhological Contexts of Language*, pages 225–243. Lawrence Erlbaum Associates, 1980.

[38] Mark Steedman. Structure and intonation. *Language*, 67:260–296, 1991.

[39] Akikazu Takeuchi and Katashi Nagao. Communicative facial displays as a new conversational modality. In *ACM/IFIP INTERCHI'93*, Amsterdam, 1993.

6 Research Acknowledgments

This research is partially supported by NSF Grant IRI91-17110, CISE Grant CDA88-22719, NSF graduate fellowships, NSF VPW GER-9350179; ARO Grant DAAL03-89-C-0031 including participation by the U.S. Army Research Laboratory (Aberdeen); U.S. Air Force DEPTH contract through Hughes Missile Systems F33615-91-C-000; DMSO through the University of Iowa; National Defense Science and Engineering Graduate Fellowship in Computer Science DAAL03-92-G-0342; and NSF Instrumentation and Laboratory Improvement Program Grant USE-9152503.

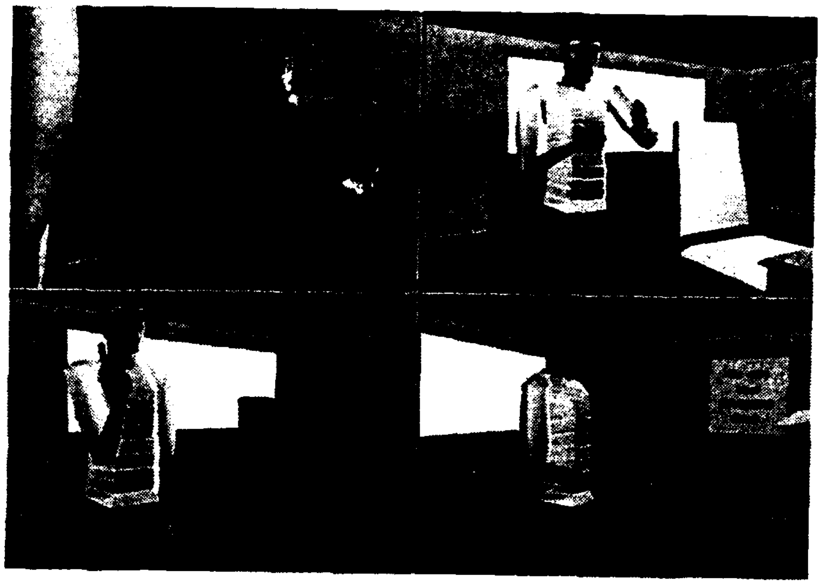


Figure 1: 4 frames representing 4 different gestures.

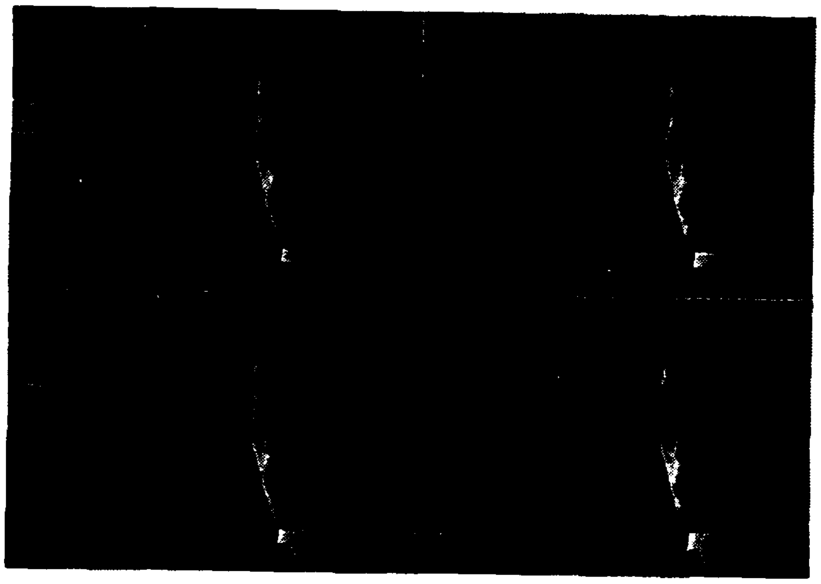


Figure 2: 4 frames representing the two agents in close up.

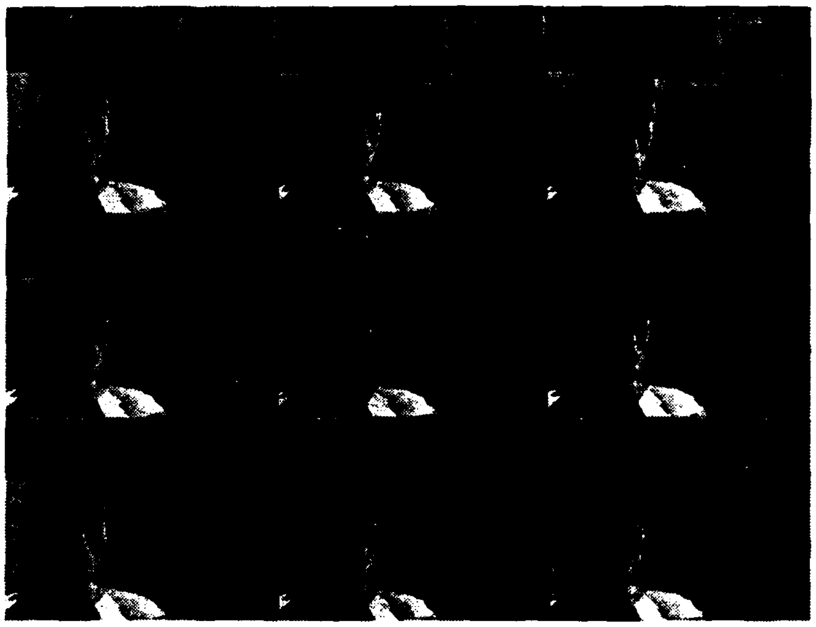


Figure 3: 9 frames representing the two agents in close up.

F Texture Resampling While Ray-Tracing: Approximating the Convolution Region Using Caching: Jeffry S. Nimeroff, Norman I. Badler, and Dimitri Metaxas

Texture Resampling While Ray-Tracing: Approximating the Convolution Region Using Caching*

Jeffry S. Nimeroff, Norman I. Badler, and Dimitri Metaxas
Computer Graphics Research Laboratory
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, Pa 19104-6389

January 25, 1994

Abstract

We present a cache-based approach to handling the difficult problem of performing visually acceptable texture resampling/filtering while ray-tracing. While many good methods have been proposed to handle the error introduced by the ray-tracing algorithm when sampling in screen space, handling this error in texture space has been less adequately addressed. Our solution is to introduce the Convolution Mask Approximation Module (CMAM). The CMAM locally approximates the convolution region in texture space as a set of overlapping texture triangles by using a texture sample caching system and ray tagging. Since the caching mechanism is hidden within the CMAM, the ray-tracing algorithm itself is unchanged while achieving an adequate level of texture filtering (area sampling as opposed to point sampling/interpolation in texture space). The CMAM is easily adapted to incorporate prefiltering methods such as MIP mapping and summed-area tables as well as direct convolution methods such as elliptical weighted average filtering.

Keywords: Ray-Tracing; Texture Resampling; Antialiasing; Filtering; Convolution; Algorithms

*This research is partially supported by ARO Grant DAAL03-89-C-0031 including participation by the U.S. Army Research Laboratory (Aberdeen), Natick Laboratory, and NASA Ames Research Center; U.S. Air Force DEPTH contract through Hughes Missile Systems F33615-91-C-0001; DMSO through the University of Iowa; NSF Grant IRI91-17110, CISE Grant CDA88-22719, and Instrumentation and Laboratory Improvement Program Grant #USE-9152503.

1 Introduction

Texture resampling is a well researched area of computer graphics. Adequate methods exist for handling the introduction of aliasing errors while "shrinkwrapping" a digital image onto the surface of a computer-generated object[6, 4, 13, 14, 18, 33, 11, 20, 17, 15]. Also, there exists a codification of the steps needed to perform the process in an "ideal" (alias-free) manner[22, 24]. These techniques rely on area information from the rendering algorithm in order to perform their function and the assumption is made that this information is readily available. The renderer is expected to provide the pixel boundary in screen space and the compound mapping (τ) from texture to screen space (surface parameterization combined with the view and screen projection). With this information the filtering module can calculate the pixel's extent in texture space (via inverse projection) and perform filtering within this extent. Exactly how each filtering method uses this information is case dependent, but all the methods referenced require some notion of the pixel's inverse projection into texture space.

Ray-tracing research has given us the ability to accomodate many optical phenomena easily within a computer-modelled environment[32, 8, 7, 19, 28]. A problem exists, however, in that the ray-tracing renderer neither explicitly computes the pixel boundary in screen space. nor explicitly constructs the compound mapping (the screen projection is replaced by the geometric ray intersection process). This appears to preclude using the well-established texture filtering algorithms without changing them severely or compromising the simplicity of the ray-tracer.

We introduce and develop the Convolution Mask Approximation Module (CMAM) and show how this simple caching module and ray tagging system can be used to create an approximation to the texture space filter extent (convolution region) that allows the ray-tracer to perform texture filtering without affecting its inherent simplicity. It will also be shown that this process adds only $O(1)$ volume (time $\times$ space) complexity to the cost associated with any of the adapted texture filtering methods. We conclude with examples of how to use the CMAM in conjunction with MIP maps[33], summed-area tables[11], and the EWA filtering technique[20].

2 Applying Textures while Ray-Tracing

Applying textures while ray-tracing can be thought of as a multi-criteria sampling process. Since only a finite number of rays can be cast for any image, aliasing in screen space is always a concern - for example, undersampling the screen space image function can allow objects to "fall between the cracks." In addition, the presence of texture mapping means that the sample locations will be used to acquire texture space information. Due to the projective native of the ray tracing "camera" geometry and the nonlinearity of many explicit surface parameterizations, samples that are well placed in screen space are not necessarily well placed in object space or in texture space (Figure 1), therefore, essential texture information maybe missed. Neither area sampling nor increasing the sampling rate solves the problem.

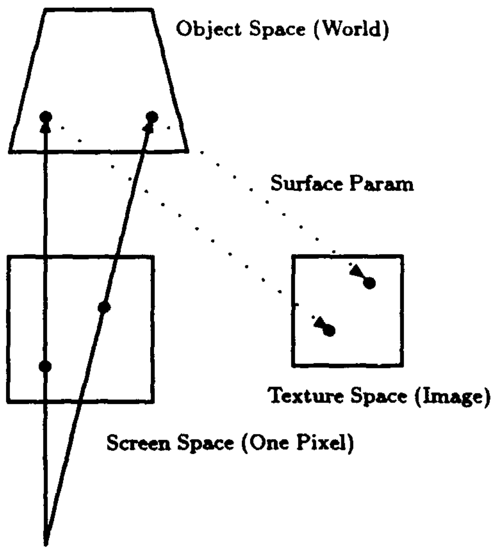


Figure 1: Dense Screen Space Distribution: Sparse in Object and Texture Space

Area sampling requires performing exact integration over the spatial extent of the projected pixel. No information is lost as with point sampling, but performing the integration is expensive, if not intractable. Two early attempts were made at performing this type of area sampling in a ray-style renderer: cone-tracing[1] and beam-tracing[23]. Cone-tracing treats each ray as a cone emanating from the chosen point and having a divergence angle. Beam-tracing projects a bundle of rays as a polygonal beam into the scene along the direction that the infinitesimally thin ray would travel. Both methods require many limiting assumptions to be made about the environment in order to remain tractable. These limitations drastically effect the usefulness of the technique.

Modifying the sampling rate (number of rays processed), using statistically significant samples in an attempt to adequately sample in screen or texture space, can minimize the affects of aliasing energy but does not remove the energy[12, 25, 7, 28]. Since a good portion of the ray-tracer's running time can be attributed to intersection calculations[32] adding extra samples can significantly affect a ray-tracer's performance. It turns out that many computer graphics textures require an infinite sampling rate to be sampled adequately.

Rather than modifying the sampling rate or allowing the limiting assumptions of an area sampling ray-style renderer, our solution follows from texture filtering research by using a modified point/area sampling method based on a local set of known texture locations. This requires keeping a window of information on the texture sampling pattern for each textured object.

3 Constructing the Convolution Region in Texture Space

The pixel's texture space extent (convolution region) is constructed by projecting the pixel's boundary points into texture space (Figure 2). A ray-tracing algorithm could do this by firing rays through the corners of the pixel and then mapping the intersection points via the surface parameterization. This

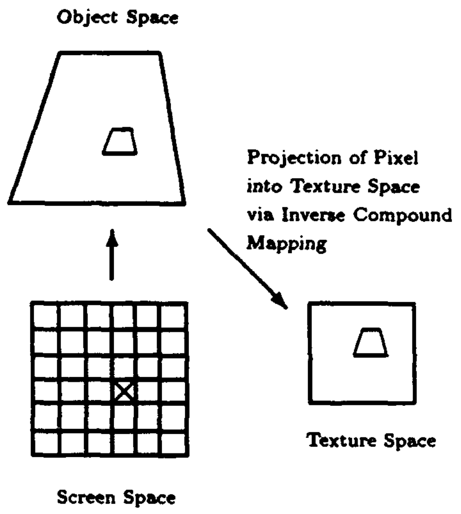


Figure 2: Convolution Region: Pixel Projection

solution suffers from limitations[1, 23] due to the coupling of the rays and also precludes using any of the simple, stochastic approaches to screen-space antialiasing[12, 25, 7, 28]. The rays can still be treated independently if one is willing to redefine the manner in which the convolution region is defined (constructed).

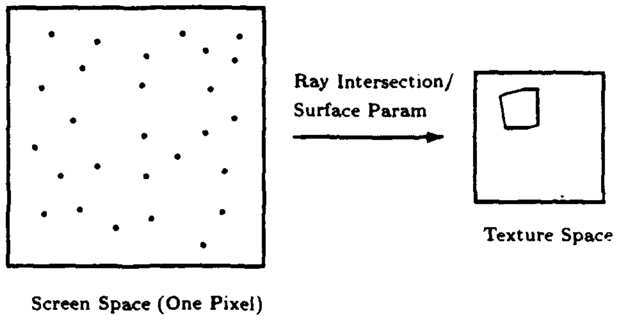


Figure 3: Convolution Region: Convex Hull of Texture Point Samples

If we modify the definition of the convolution region to include that area of texture space inside the convex hull of a set of texture space point samples (Figure 3), an incremental approach to texture filtering while ray-tracing evolves.

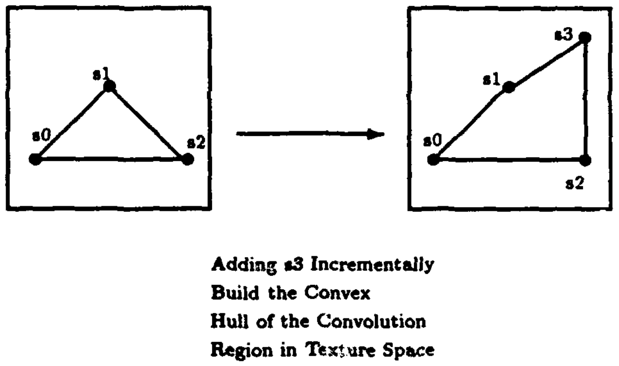


Figure 4: Incremental Construction of the Convolution Region

Our incremental convex hull filtering method approximates the filtering region in texture space as a set of (possibly) overlapping texture triangles (Figure 4). The current sample location along with the two previous sample locations (provided the rays emanate from the same pixel) are used to give a local approximation to the texture area that needs to be filtered for this sample. This overestimates the convolution region by allowing for the inclusion of a texture sample more than once, but guarantees that only those samples inside the convex hull are included in the filtering operation. A non-incremental approach is not as useful because every point sample of texture space cannot easily/accurately be associated with a filtered texture intensity (filter values are only associated with areas incrementally bound by the point samples).

3.1 Caching and the Convolution Mask Approximation Module

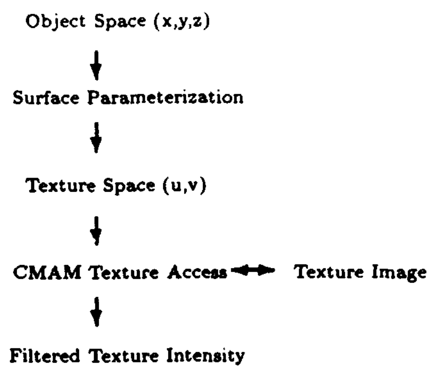


Figure 5: Placement of the Convolution Mask Approximation Module

The convolution mask approximation module (CMAM) is a data structure and a set of routines that resides between the texture image (or data structure) and the surface parameterization (Figure 5) and implements the methodology described above.

```
typedef float Color[COLOR_SPACE];

typedef struct cmam {
    /* Flag for turning CMAM filtering on */
    int filter;

    /* Ray IDs of cached samples */
    int last_id, sec_last_id;

    /* Recursion levels of cached samples */
    int last_level, sec_last_level;

    /* Sample (u,v) of cached samples */
    float last_u, sec_last_u;
    float last_v, sec_last_v;

    /* Texture data structure */
    Color **map;

    /* Bounds of texture array */
    int rows, cols;
} Cmam, *Cmamptr;
```

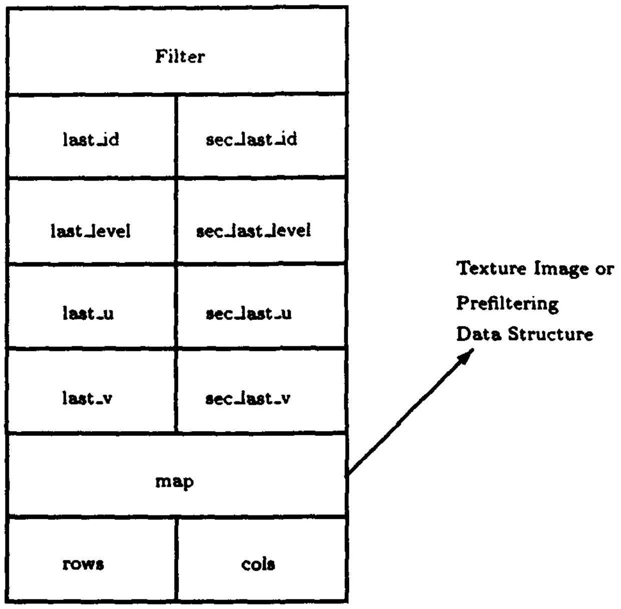


Figure 6: CMAM and Associated Texture Data Structure

Instead of having the ray-tracer accessing the texture image directly and performing filtering itself, the CMAM takes the texture location, and returns the filtered texture value to the ray-tracer.

The convolution region is approximated as above with ray tagging being used to facilitate the process of finding related rays. Rays which are fired through pixels on the same scanline in screen space, and might ultimately be used to bound a region in texture space, are given the same ID and a starting recursion level of zero. IDs, levels, and sample locations are passed to the CMAM which compares them with the most recent CMAM accesses. ID matching facilitates the incremental building of the convolution area along the scanline. Level matching allows the accumulation of texture area information treating proxi-

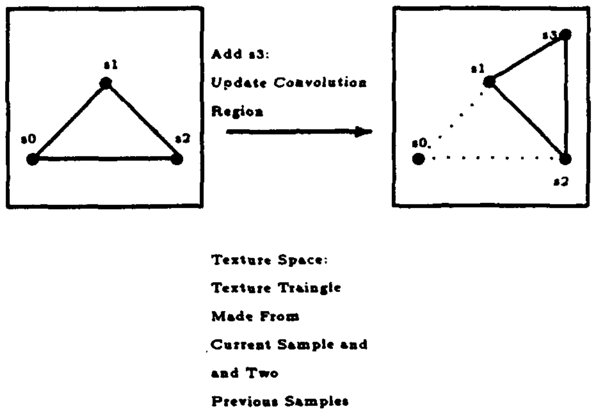


Figure 7: CMAM Approximation of Overlapping Texture Triangles

mate groups of reflected/refracted rays as an approximation to the travelling wavefront[1, 23]. If the IDs and levels match then the region bounded by the samples in texture space is part of an approximation to the true convolution region.

The CMAM then can use known filtering techniques such as MIP mapping, summed-area tables or an EWA scheme to filter the area and return a texture intensity (without any interdependence of the rays). No changes to the basic ray-tracing implementation are necessary. The ray-tracer acts as if it is point sampling in texture space.

3.2 Using the CMAM with Existing Techniques

How the CMAM uses its local approximation of the convolution region is specific to the type of texture filtering that is going to be performed. MIP mapping, summed-area tables, and EWA filtering require their convolution regions to be described in different ways. When the texture sample is sent to the CMAM the ID is checked against the cached values. This current ID can match the IDs of both the cached samples, the ID of the most recently sample only, or any of the IDs in the cache. The problem then reduces to generating an area based on the number of cache hits counted and the texture space sampling pattern.

3.2.1 MIP Map Approximation

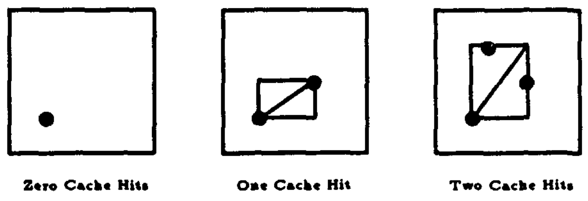


Figure 8: MIP Map Approximation Using the CMAM

Williams' MIP mapping[33] performs texture filtering by accessing a pre-filtered texture pyramid and performing trilinear interpolation. The pyramid is accessed with a d parameter which chooses the two levels which best approximate the filtered region. Intra-level access is via the texture coordinate (u, v) and uses bilinear interpolation to reconstruct the texture value within the levels. The only hard part seems to be constructing d using the CMAM.

We us the following MIP map level approximation algorithm.

1. If the sample does not match the most recent sample, use the highest level of the pyramid (the average intensity of the texture image) to trade blurring for aliasing (texture image with high frequencies), or use the lowest level of the pyramid (point sample) to trade aliasing for blurring (texture image with high frequencies).

   $$d = 0 \quad \text{or} \quad d = MAXLEVEL$$

   Point sampling trades aliasing for blurring, while using the fully averaged texture image trades blurring for aliasing. Since the human visual sustem is more tolerent of blurring than aliasing, we chose to use the averaged texture image.

2. If the sample matches the most recent sample use the length of the line between the two samples in the following calculation:

   $$d = \lg(\text{length of line between samples})$$

3. If the sample matches the both cached samples, fit an axis-aligned bounding box around the three samples. Use the length of the diagonal of this bounding box in the same calculation as above.

   $$d = \lg(\text{length of diagonal of bounding box})$$

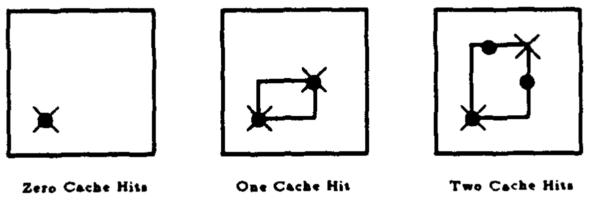


Figure 9: Summed-Area Table Approximation Using the CMAM

3.2.2 Summed-Area Table Approximation

The summed-area table[11] is not restricted to filtering square regions in texture space. It is accessed using the corners of the axis-aligned rectangular region that is to be convolved with a box or Gaussian filter. The summed-area table access is simple for the CMAM.

1. If the sample does not match the most recent sample, use the texture space coordinate to either point sample or average the region from the origin of the summed-area table to the texture coordinate (same criteria as mentioned above).

2. If the sample matches the most recent sample use the two texture space coordinates to create a rectangular region to be filtered using the summed-area table.

3. If the sample matches the both the cached samples, fit an axis-aligned bounding box around the three samples. Use the upper-right and lower-left coordinates to access the summed-area table.

3.2.3 EWA Approximation

The EWA algorithm[20] is a direct convolution algorithm (not prefiltering) and requires the semi-major and semi-minor axes of a texture space elliptical filtering region as well as a warped filter function. Since the ray-tracer lacks the inverse compound mapping τ^{-1}, computing the warped filter function can only be approximated. The axes for the texture space ellipse and the filter kernel access can be done as follows:

1. No matches uses point sampling as with MIP maps or summed-area tables.

2. If the sample matches the most recent sample use the two texture space coordinates to create a line which represents the radius of a texture space

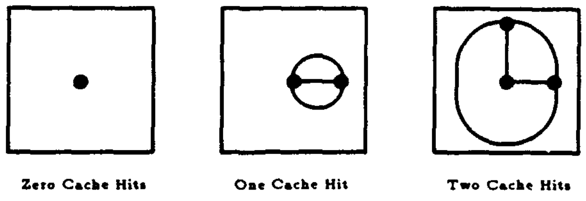


Figure 10: EWA Approximation Using the CMAM

circle (degenerate ellipse). For each texture sample contained within the circle, use its distance from the center to access a circular symmetric filter kernal and weight the sample accordingly.

3. If the sample matches both the cached samples, use the vector from the current sample to one of the cached samples that is the longest as the semi-major axis of he texture space ellipse and the vector between the current point and the other cached sample as the semi-minor axis of the ellipse. The circularly symmetric filter kernel can then be accessed by the distance from the center of the ellipse to the sample, normalized by the distance from the center of the ellipse to the boundary of the ellipse that runs through the sample point and its value used to weight the samples within the texture space ellipse.

4 Convex Hull Weighting

Since the CMAM caches only the last two texture samples it is possible that the area bounding the three most recent samples overlaps a region of the texture image that already has been included in the final filtered intensity for the current pixel. We have begun to investgate a method for incorporating the convex hull of the union of all the approximated convolutions into the CMAM. When a region is to be filtered, the region is differenced with the convex hull to return that part of the convolution region which has not yet been incorporated into the filtered texture intensity (the DC value of the texture image is returned if the current region is completely enclosed within the convex hull of the approximate convolution region). This type of weighting will include each element of the texture image (within the true convolution region) at most once in the final filtered intensity, thereby yielding a better approximation to the true texture intensity.

5 Conclusion

The convolution mask approximation module provides a caching system which is useful for approximating the texture space extent of the screen space filter kernel. It uses ray coherence to locally approximate the filter extent as a set of triangular regions that tile areas of the true convolution region. Filtering then becomes possible using these areas to access either prefiltered data structures or a direct convolution filtering algorithm[33, 11, 20]. With the use of the CMAM we are able to perform texture filtering without changing the simplicity of the basic ray-tracing implementation or limiting any of its photo-realistic features.

References

[1] John Amanatides. Ray Tracing with Cones. *Computer Graphics*, 18(3):129–135, July 1984.

[2] James F. Blinn. Return of the Jaggy. *IEEE Computer Graphics ana Applications*, 9(2):82–89, March 1989.

[3] James F. Blinn. What We Need Around Here Is More Aliasing. *IEEE Computer Graphics and Applications*, 9(1):75–79, January 1989.

[4] James F. Blinn and Martin E. Newell. Texture and Reflection in Computer-Generated Images. *Communications of the ACM*, 19(10):542–547, October 1976.

[5] Ronald Bracewell. *The Fourier Transform and its Applications*. McGraw-Hill, 1986.

[6] Edwin A. Catmull. Computer Display of Curved Surfaces. *Proc. Conf. on Computer Graphics, Pattern Recognition, Data Structure*, pages 11–17, May 1975.

[7] Robert L. Cook. Stochastic Sampling in Computer Graphics. *ACM Transactions on Graphics*, 5(1):51–72, Januray 1986.

[8] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed Ray Tracing. *Computer Graphics*, 18(3):139–147, July 1984.

[9] Franklin C. Crow. The Aliasing Problem in Computer-Generated Shaded Images. *Communications of the ACM*, 20(11):799–805, November 1977.

[10] Franklin C. Crow. A Comparison of Antialiasing Techniques. *IEEE Computer Graphics and Applications*, 1(1):40–47, January 1981.

[11] Franklin C. Crow. Summed-Area Tables. *Computer Graphics*, 18(3):207–212, July 1984.

[12] Mark A. Z. Dippé and Erling Henry Wold. Antialiasing Through Stochastic Sampling. *Computer Graphics*, 19(3):69–78, July 1985.

[13] William Dungen Jr., Anthony Stenger, and George Sutty. Texture Tile Considerations for Raster Graphics. *Computer Graphics*, 12(3):130–134, August 1978.

[14] Eliot A. Feibush, Marc Levoy, and Robert L. Cook. Synthetic Texturing Using Digital Filters. *Computer Graphics*, 14(3):294–301, July 1980.

[15] Eugene Fiume and Robert Lansdale. Fast Space-Variant Texture Filtering Techniques. *SPIE 1991 Proceedings*, 1991. Preliminary version of the paper submitted for publication.

[16] James D. Foley, Andries van Dam, Steven Feiner, and John Hughes. *Computer Graphics - Principles and Practice*. Addison-Wesley, 1990.

[17] Alain Fournier and Eugene Fiume. Constant-Time Filtering with Space-Variant Kernals. *Computer Graphics*, 22(4):229–237, August 1988.

[18] Michel Ganget, Didier Perny, and Phillippe Coueignoux. Perspective Mapping of Planar Textures. *Computer Graphics*, 16(1), May 1982.

[19] Andrew Glassner. *An Introduction to Ray Tracing*. Academic Press, New York, 1989.

[20] Ned Greene and Paul Heckbert. Creating Raster Omnimax Images from Multiple Perspective Views Using the Elliptical Weighted Average Filter. *IEEE Computer Graphics and Applications*, pages 21–27, June 1986.

[21] Paul Heckbert. Survey of Texture Mapping. *IEEE Computer Graphics and Applications*, 6(11):56–67, November 1986.

[22] Paul S. Heckbert. Fundamentals of Texture Mapping and Image Warping. Master's thesis, Division of Computer Science, University of California at Berkeley, June 1989.

[23] Paul S. Heckbert and Pat Hanrahan. Beam Tracing Polygonal Objects. *Computer Graphics*, 18(3):119–128, July 1984.

[24] Robert C. Lansdale. Texture Mapping and Resampling for Computer Graphics. Master's thesis, Department of Electrical Engineering, University of Toronto, January 1991.

[25] Mark E. Lee, Richard A. Redner, and Samuel P. Uselton. Statistically Optimized Sampling for Distributed Ray Tracing. *Computer Graphics*, 19(3):61–67, July 1985.

[26] Dimitri Metaxas and Evangelos Milios. Reconstruction of a Color Image from Nonuniformly Distributed Sparse and Noisy Data. *CVGIP: Graphics Models and Image Processing*, pages 103–111, March 1992.

[27] Don P. Mitchell. Generating Antialiased Images at Low Sampling Densities. *Computer Graphics*, 21(4):65–69, July 1987.

[28] Don P. Mitchell. The Antialiasing Problem in Ray Tracing. *Advanced Topics in Ray Tracing, SIGGRAPH 1990 Course Notes*, 1990.

[29] Don P. Mitchell and Arun N. Netravali. Reconstruction Filters in Computer Graphics. *Computer Graphics*, 22(4):221–228, August 1988.

[30] Donald E. Pearson. *Transmission and Display of Pictorial Information*. Halsted Press, John Wiley & Sons, New York, 1975.

[31] Ken Shoemake. Personal communication, January 1992.

[32] Turner Whitted. An Improved Illumination Model for Shaded Displays. *Communications of the ACM*, 23(6), June 1980.

[33] Lance Williams. Pyramidal Parametrics. *Computer Graphics*, 17(3):1–11, July 1983.

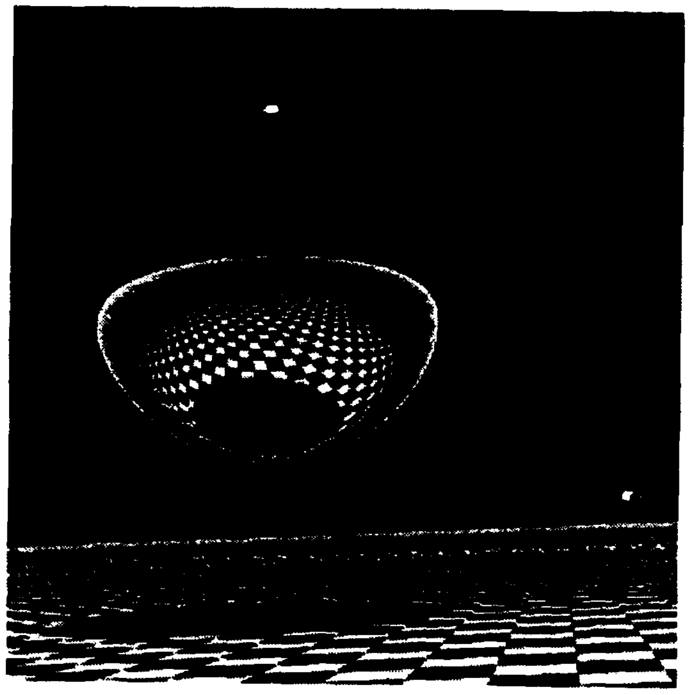


Figure 11: Test Scene 1 with CMAM on

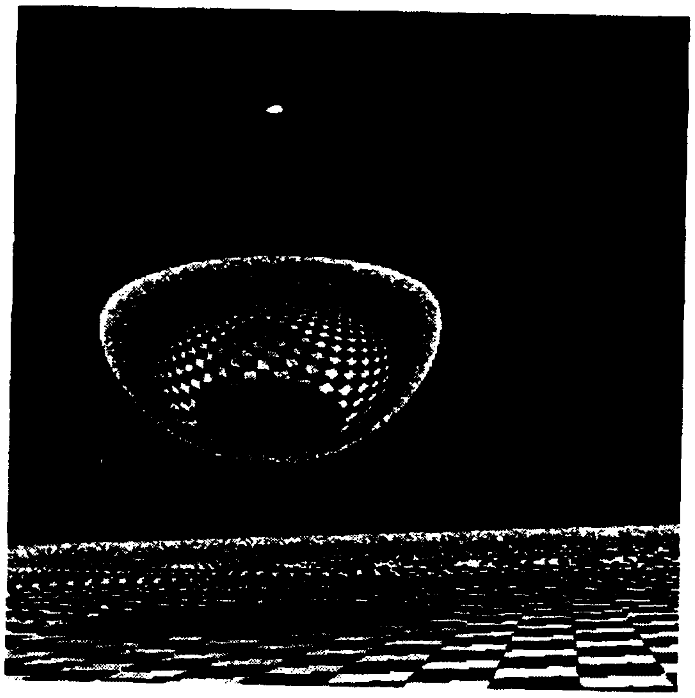


Figure 12: Test Scene 1 with CMAM off

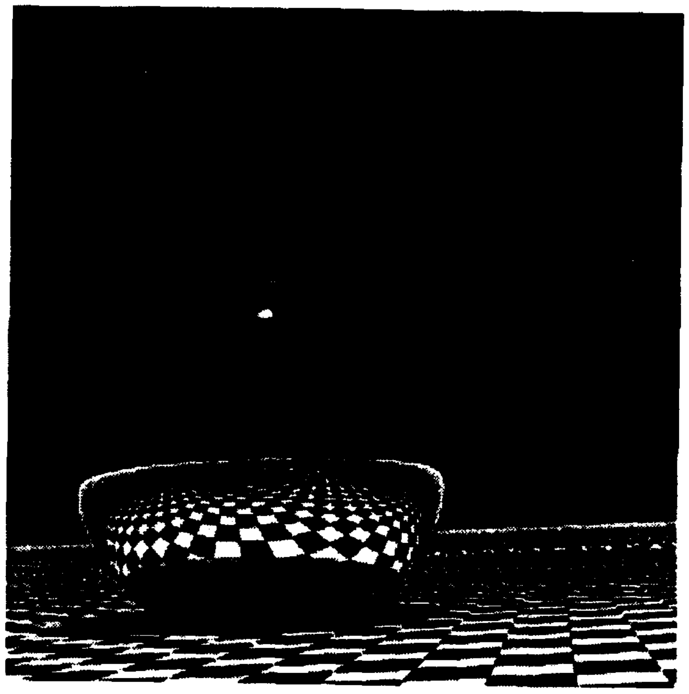


Figure 13: Test Scene 2 with CMAM on

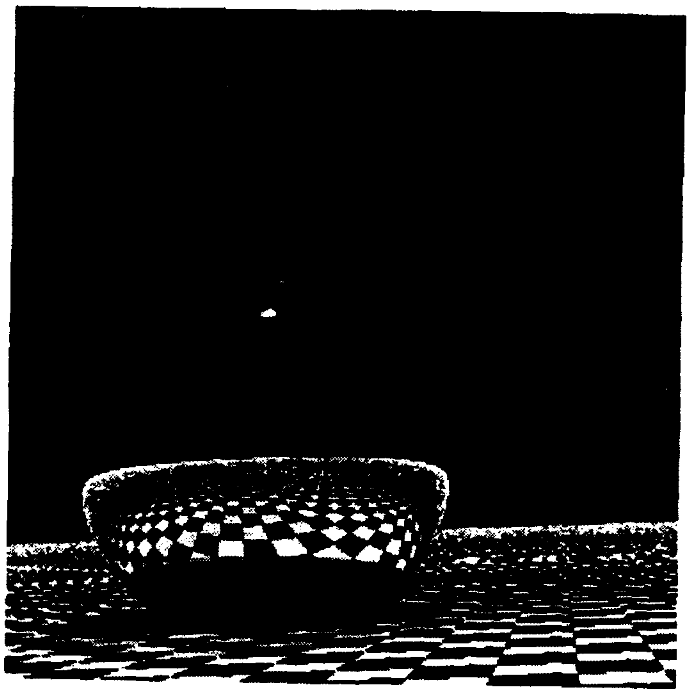


Figure 14: Test Scene 2 with CMAM off

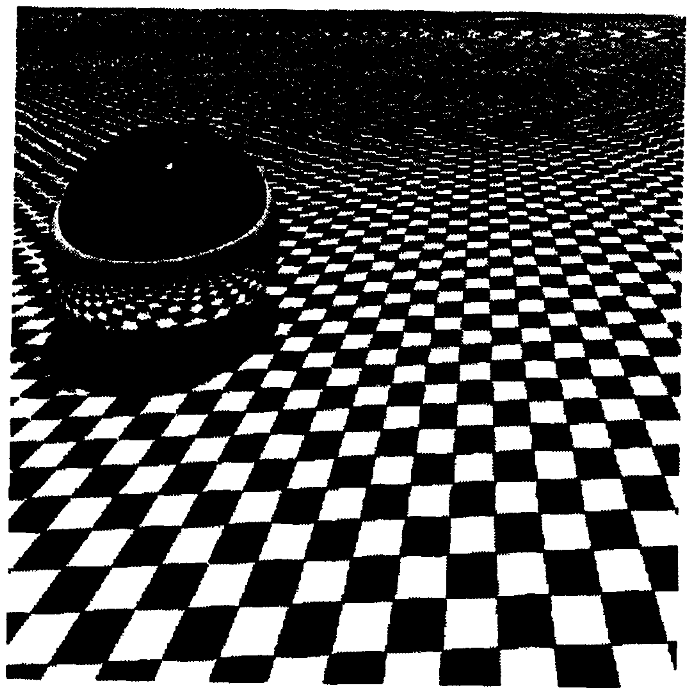


Figure 15: Test Scene 3 with CMAM on

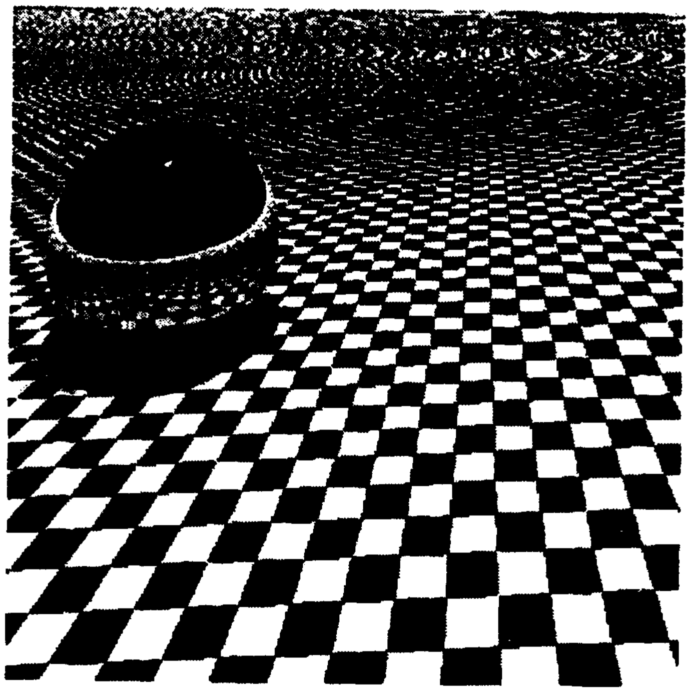


Figure 16: Test Scene 3 with CMAM off

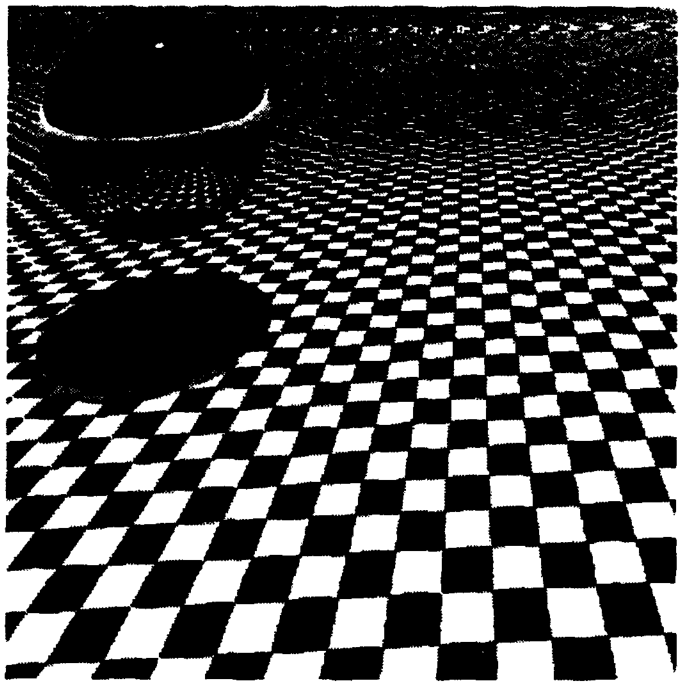


Figure 17: Test Scene 4 with CMAM on

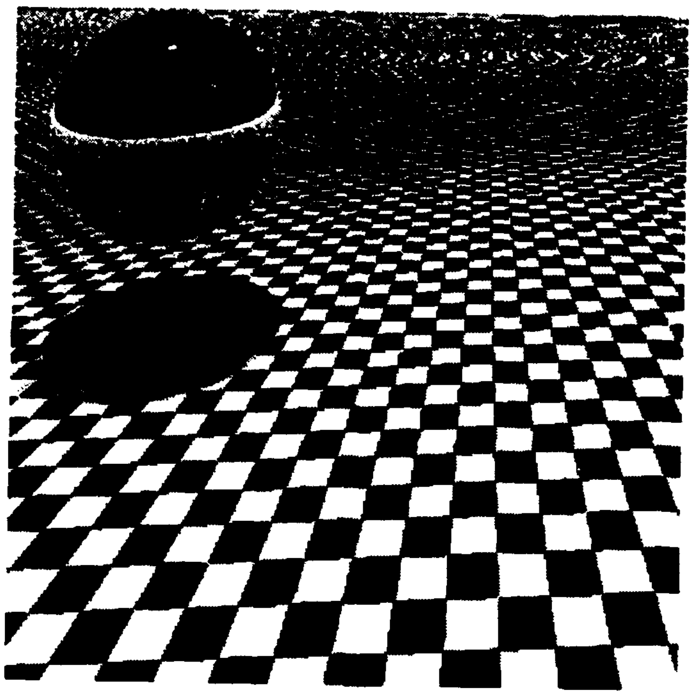


Figure 18: Test Scene 4 with CMAM off

G Dynamic Fluid Simulations: Waves, Splashing, Vorticity, Boundaries, Buoyancy: Nick Foster and Dimitri Metaxas

Dynamic Fluid Simulations: Waves, Splashing, Vorticity, Boundaries, Buoyancy

Nick Foster and Dimitri Metaxas
Department of Computer and Information Science,
University of Pensylvania, Philadelphia, PA 19104

Abstract

This paper develops a new general approach for the modeling and animation of viscous incompressible fluids. We use the full time-dependent Navier-Stokes equations to comprehensively simulate 2D and 3D incompressible fluid phenomena. Unlike previous fluid animation graphics techniques which were based on approximating solutions to fluid motion, the use of these equations accurately models shallow or deep fluid flow, transient dynamic flow, vorticity and splashing in simulated physical environments. In our simulations we can also include variously shaped and spaced static or moving obstacles that are fully submerged or penetrate the fluid surface. We use stable numerical analysis techniques based on finite-differences for the solution of the Navier-Stokes equations. We solve these equations by iterating an initial set of pressures and velocities defined over the finite-difference grid. To model free-surface fluids, we present a powerful technique which is based on the Marker-and-Cell method. Based on the fluid's pressure and velocities obtained from the solution of the Navier-Stokes equations this technique allows modeling of the fluid's free surface either by solving a surface equation or by tracking the motion of marker particles. The later technique is suitable for visualization of splashing and vorticity. Furthermore, we develop an editing tool for easy definition of a physical-world which includes obstacles, boundaries and fluid properties such as viscosity, initial velocity and pressure. Using our editor we can perform complex fluid simulations without prior knowledge of the underlying fluid dynamic equations. Finally, depending on the application we render the fluid's surface using well-developed graphics techniques and standard SGI workstation hardware routines.

Category: Research paper, **Format:** Regular paper.

Keywords: Dynamic Fluid Simulations, Physics-Based Modeling, Transient Dynamics, Free-surface Flow, Navier-Stokes Equations, Boundaries, Buoyancy, Finite Difference.

0

1 Introduction

Modeling realistically and accurately scenes involving fluid simulations and especially water for graphics applications is a challenging problem that has captured the attention of several researchers [17, 10, 15, 5, 19, 11, 8]. These techniques whether suitable for still images or animations are characterized by the use of approximating techniques for simulating fluids. Furthermore, they model a restricted range of fluid phenomena. Among others, realistic and accurate, splashing, vorticity, three-dimensional flows, transient fluid dynamics, static and dynamic obstacles can't be handled. Modeling of the above phenomena and related fluid effects are becoming increasingly necessary in fluid animations. This paper addresses the above problems based on the use of the Navier-Stokes differential equations to model fluid motion. Furthermore, through the use of numerical analysis techniques the computational time required for the complex simulations in this paper is between fifteen minutes to two hours on a standard Silicon Graphics Crimson workstation.

In [11] and [9] techniques for producing still images based on Fourier synthesis and stochastic subdivision, respectively, were developed. Among the first methods for fluid animation were techniques for modeling waves [10, 17, 16] in deep water away from various boundaries. In [15, 5, 19] more sophisticated techniques for wave modeling were developed due to the introduction of refraction resulting in changes to wave velocity with depth. The basic idea behind these papers was the use of particles moving in a restricted way (circular or ellipsoidal) around their initial positions. As a result, even though they could simulate phenomena like waves hitting a beach they could not handle more complex animations like wave reflections, changes in the topology of the water and raindrops hitting surfaces. In order to overcome the above limitations, Kass and Miller [8] introduce a simple and very efficient technique for simulating water based on the solution of a set of partial differential equations which approximate the shallow water equations. Through their method, the water surface is represented as a height field which evolves over time based on the integration of the above shallow water differential equations. Finally, to reduce the complexity of three dimensional water animations, they approximate the three-dimensional equations with a series of two-dimensional shallow water equations. Methods based on particles systems [13, 21] have also been used for approximating the modeling of wave propagation. Their main limitations are the lack of realistic three dimensional wave animations, ease of surface rendering, instability, control and difficulty to simulate correctly vorticity, even if a large number of them is used. All of the above graphics techniques for simulating fluid phenomena are based on simplified fluid models which are sufficient for a limited set of fluid simulations described above. There is an increasing demand though in fluid animations for more realistic and pleasing animations, modeling of complex phenomena such as vorticity and splashing, and inclusion of static and dynamic objects in fluid simulations.

In this paper we develop a new general approach for modeling and animation of a superset of fluid phenomena compared to previous techniques. We use the full time-dependent Navier-Stokes equations [4] to comprehensively simulate 2D and 3D incompressible fluid phenomena. These equations accurately model shallow or deep fluid flow, transient dynamic flow, vorticity and splashing in simulated physical environments. In our simulations we can also include variously shaped and spaced static or moving obstacles that are fully submerged, penetrate or float on the fluid surface. We use stable numerical analysis techniques based on finite-differences for the solution of the Navier-Stokes equations. We solve these equations by iterating an initial set of pressures and velocities defined over the finite-difference grid. Our technique does not require the solution of large linear systems at every iteration making the solution significantly more efficient to standard ways of solving the above equations.

To model free-surface fluids, we present a technique which is based on the Marker-and-Cell method [6, 4]. Based on the fluid's pressure and velocities obtained from the solution of the Navier-Stokes equations this technique allows modeling of the fluid's free surface either by solving a differential equation for the surface's height or by tracking the motion of marker particles. Marker particles follow the fluid according to the velocity components in their vicinity obtained from the solution of the Navier-Stokes equations. We use them only for the purpose of indicating fluid configuration and they do not participate in the velocity and pressure calculation. They show which cells in the finite-difference grid contain fluid and especially which cells lie along the free surface. They also serve as a flow visualization coordinate system whereby fluid element trajectories and relative positions can be observed. Therefore, we use them for visualizing splashing and vorticity within a fluid.

Furthermore, we develop an editing tool for easy definition by a user of a physical-world which includes obstacles, boundaries and fluid properties such as viscosity, initial velocity and pressure. Using our editor we can then perform complex fluid simulations without knowledge of the underlying fluid dynamic equations. Finally, depending on the application we render the fluid's surface using well-developed graphics techniques and standard SGI workstation hardware routines.

In the following sections we present the details of our technique and demonstrate its power with a number of experiments.

2 Navier-Stokes Equations

The Navier-Stokes equations for viscous incompressible fluids are the following set of three-dimensional differential equations which model the conservation of momentum

$$\frac{\partial u}{\partial t} + \frac{\partial u^2}{\partial x} + \frac{\partial uv}{\partial y} + \frac{\partial uw}{\partial z} = -\frac{\partial p}{\partial x} + g_x + \nu(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}),$$
$$\frac{\partial v}{\partial t} + \frac{\partial vu}{\partial x} + \frac{\partial v^2}{\partial y} + \frac{\partial vw}{\partial z} = -\frac{\partial p}{\partial y} + g_y + \nu(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2}),$$
$$\frac{\partial w}{\partial t} + \frac{\partial wu}{\partial x} + \frac{\partial wv}{\partial y} + \frac{\partial w^2}{\partial z} = -\frac{\partial p}{\partial z} + g_z + \nu(\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial z^2}),$$
$$(1)$$

and mass
$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \qquad (2)$$

in fluid flow. p is the ratio of pressure to constant density, g_x, g_y, g_z, are prescribed body accelerations (e.g., gravity), ν is the coefficient of kinematic viscosity, and u, v, w are the fluid velocities in the x, y and z dimensions, respectively.

3 Discretization

There are several ways to solve numerically those complex nonlinear differential equations [4]. In this paper we use a finite-difference approximation to these equations called the Marker and Cell method [4] due to its simplicity and suitability for free surface modeling. In this approach we divide the region in which we do the computations into a set of small rectangular cells having edge lengths δx, δy, and δz. With respect to this set of computational cells, we define velocity components on cell faces and pressure values at cell centers as shown in Fig. 1. We label cells with an index (i, j, k), which denotes the cell number as counted from the origin in the x, y, and z directions, respectively. Also $p_{i,j,k}$ is the pressure at the center of cell (i, j, k), while $u_{i+1/2,j,k}$ is the x-direction velocity at the center of the face between cells (i, j, k) and $(i + 1, j, k)$. We number similarly the velocities in the y and z directions.

We obtain a a time dependent solution by advancing the flow field variables through a sequence of short time steps of duration δt. We then break the calculation of the advancement for one step in two stages. First we update all the velocity components using the previous state of the flow to calculate the accelerations caused by convection, viscous stresses, body forces, etc. This first stage consists of an explicit velocity calculation. This computation though does not necessarily lead to a velocity field with zero divergence, that is to one that conserves mass. Therefore, in stage two adjustments must be made to insure mass conservation. We do so by adjusting the pressure in each cell in such a way that there is no net mass flow in or out of the cell. A change in one cell will affect neighboring cells so that this pressure adjustment must be performed iteratively until all cells have simultaneously achieved a zero mass change. In the second stage we adopt a technique developed by Chorin [3] in which a simultaneous

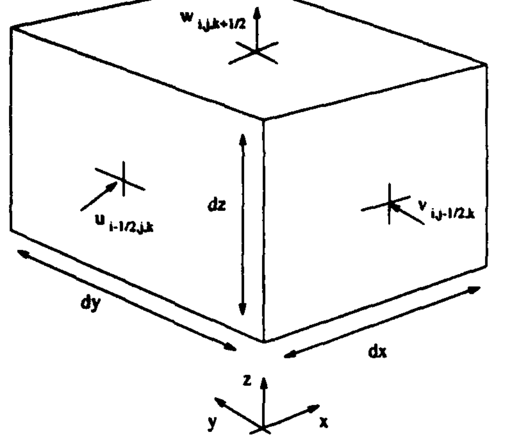


Figure 1: Location of velocity components on a typical cell (i, j, k).

iteration on pressures and velocities is performed. This technique simplifies the application of boundary conditions.

3.1 Stage 1

We advance explicitly the velocities in stage one, which results in quantities labeled by tildes based on the following formulas which are a finite-difference approximation to (1).

$$\begin{aligned}
\tilde{u}_{i+1/2,j,k} &= u_{i+1/2,j,k} + \delta t\{(1/\delta x)[(u_{i,j,k})^2 - (u_{i+1,j,k})^2] \\
&+(1/\delta y)[(uv)_{i+1/2,j-1/2,k} - (uv)_{i+1/2,j+1/2,k}] \\
&+(1/\delta z)[(uw)_{i+1/2,j,k-1/2} - (uw)_{i+1/2,j,k+1/2}] + g_x \\
&+(1/\delta x)(p_{i,j,k} - p_{i+1,j,k}) + (\nu/\delta x^2)(u_{i+3/2,j,k} \\
&-2u_{i+1/2,j,k} + u_{i-1/2,j,k}) + (\nu/\delta y^2)(u_{i+1/2,j+1,k} \\
&-2u_{i+1/2,j,k} + u_{i+1/2,j-1,k}) + (\nu/\delta z^2)(u_{i+1/2,j,k+1} \\
&-2u_{i+1/2,j,k} + u_{i+1/2,j,k-1})\}, \\
\tilde{v}_{i,j+1/2,k} &= v_{i,j+1/2,k} + \delta t\{(1/\delta x)[(vu)_{i-1/2,j+1/2,k} \\
&-(vu)_{i+1/2,j+1/2,k}] + (1/\delta y)[(v_{i,j,k})^2 - (v_{i,j+1,k})^2] \\
&+(1/\delta z)[(vw)_{i,j+1/2,k-1/2} - (vw)_{i,j+1/2,k+1/2}] + g_y \\
&+(1/\delta y)(p_{i,j,k} - p_{i,j+1,k}) \\
&+(\nu/\delta x^2)(v_{i+1,j+1/2,k} - 2v_{i,j+1/2,k} + v_{i-1,j+1/2,k}) \\
&+(\nu/\delta y^2)(v_{i,j+3/2,k} - 2v_{i,j+1/2,k} + v_{i,j-1/2,k}) \\
&+(\nu/\delta z^2)(v_{i,j+1/2,k+1} - 2v_{i,j+1/2,k} + v_{i,j+1/2,k-1})\}, \\
\tilde{w}_{i,j,k+1/2} &= u_{i,j,k+1/2} + \delta t\{(1/\delta x)[(wu)_{i-1/2,j,k+1/2} \\
&-(wu)_{i+1/2,j,k+1/2}] + (1/\delta y)[(wv)_{i,j-1/2,k+1/2} \\
&-(wv)_{i,j+1/2,k+1/2}] + (1/\delta z)[(w_{i,j,k})^2 - (w_{i,j,k+1})^2] + g_z \\
&+(1/\delta z)(p_{i,j,k} - p_{i,j,k+1}) \\
&+(\nu/\delta x^2)(w_{i+1,j,k+1/2} - 2w_{i,j,k+1/2} + w_{i-1,j,k+1/2}) \\
&+(\nu/\delta y^2)(w_{i,j+1,k+1/2} - 2w_{i,j,k+1/2} + w_{i,j-1,k+1/2}) \\
&+(\nu/\delta z^2)(w_{i,j,k+3/2} - 2w_{i,j,k+1/2} + v_{i,j,k-1/2})\}. \qquad (3)
\end{aligned}$$

We calculate the velocities and pressures needed at locations other than where they are defined as simple averages, e.g., $u_{i,j,k} = \frac{1}{2}(u_{i+1/2,j,k} + u_{i-1/2,j,k})$, and the square of a quantity, e.g., u^2 at (i, j, k) is the square of the average, $(u_{i,j,k})^2$, rather than the average of the squares, $u^2_{i+1/2,j,k}$ and $u^2_{i-1/2,j,k}$.

3.2 Stage 2

Since equations (3) are the finite difference approximation of (1), they do not necessarily result in a velocity field that satisfies (2). To ensure mass conservation we apply an iterative process in which the cell pressures are modified to make the velocity divergence vanish. In each cell (i, j, k) the value of the velocity divergence D is calculated as

$$\begin{aligned}
D_{i,j,k} &= (1/\delta x)(u_{i+1/2,j,k} - u_{i-1/2,j,k}) + (1/\delta y)(v_{i,j+1/2,k} - \\
&\quad v_{i,j-1/2,k}) + (1/\delta z)(w_{i,j,k+1/2} - w_{i,j,k-1/2}). \qquad (4)
\end{aligned}$$

If the magnitude of D is less than some prescribed small value ϵ (in this paper we used 10^{-3}), the flow is locally incompressible and no change in the cell velocity is necessary. However, if the magnitude of D is larger than ϵ then the pressure is changed by

$$\delta p = -\beta D, \qquad (5)$$

where β is given by

$$\beta = \beta_0/2\delta t(\frac{1}{\delta x^2} + \frac{1}{\delta y^2} + \frac{1}{\delta z^2}). \qquad (6)$$

The constant β_0 is a relaxation factor, where overrelaxation and underrelaxation correspond to β_0 greater than or less than unity, respectively. For iteration stability it is necessary to keep $\beta_0 < 2$. In our applications we used the value $\beta_0 = 1.6$, but the value of β_0 giving the most rapid convergence can only be determined by experimentation [4].

Once we calculate δp for a cell (i, j, k) we add it to the pressure $p_{i,j,k}$ and we adjust the velocity components on the sides of cell (i, j, k) based on the following formulas

$$\begin{aligned}
u_{i+1/2,j,k} &= u_{i+1/2,j,k} + (\delta t/\delta x)\delta p, \\
u_{i-1/2,j,k} &= u_{i-1/2,j,k} - (\delta t/\delta x)\delta p, \\
v_{i,j+1/2,k} &= v_{i,j+1/2,k} + (\delta t/\delta y)\delta p, \\
v_{i,j-1/2,k} &= v_{i,j-1/2,k} - (\delta t/\delta y)\delta p, \\
w_{i,j,k+1/2} &= w_{i,j,k+1/2} + (\delta t/\delta z)\delta p, \\
w_{i,j,k-1/2} &= w_{i,j,k-1/2} - (\delta t/\delta z)\delta p, \qquad (7)
\end{aligned}$$

We then repeat this process successively until no cell has a magnitude of D greater than ϵ. When the iteration has converged (it usually takes approximately 5 iterations) the adjusted velocities satisfy the mass-conservation equation (2) and this completes the necessary calculations for advancing the flow field through one cycle in time.

4 Free Surface Calculation

In fluid applications which include free surfaces which are single-valued (e.g., rivers, lakes, oceans, no splashing waves), the height of the free surface can be dynamically computed from the velocities computed through the Navier-Stokes equations. Through this technique the surface height (computed from the bottom of the mesh) is defined at the center of each vertical column of cells in the three-dimensional mesh. The change in the local surface elevation is determined by the local fluid velocity, that is, by the vertical component of the fluid motion plus the horizontal convection of the surface elevation from adjacent cell columns,

$$\frac{\partial h}{\partial t} = w - u(\frac{\partial h}{\partial x}) - v(\frac{\partial h}{\partial y}). \tag{8}$$

When a finite difference approximation is used, the above equation is numerically unstable due to a negative diffusion truncation error [4]. We compensate this error by adding to the surface height equation a positive diffusion term

$$\gamma(\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2}), \tag{9}$$

where γ is a constant diffusion coefficient, chosen as

$$\gamma > \frac{\delta t}{2} max(u^2, v^2), \tag{10}$$

to ensure stability. In our paper we used a value of $\gamma = 1.2$. Based on the above equations (8) and (9), the finite difference form (space-centered and forward in time) of the kinematic surface equation is

$$\begin{aligned} h_{i,j}^{t+\delta t} &= h_{i,j}^{t} + \delta t\{\bar{w}_{i,j,k}^{t+\delta t} + \frac{1}{4\delta x}(u_{i+1/2,j,k}^{t+\delta t} + u_{i-1/2,j,k}^{t+\delta t}) \\ &(h_{i-1,j}^{t} - h_{i+1,j}^{t}) + \frac{1}{4\delta y}(v_{i,j+1/2,k}^{t+\delta t} + v_{i,j-1/2,k}^{t+\delta t}) \\ &(h_{i,j-1}^{t} - h_{i,j+1}^{t}) + \gamma[\frac{1}{\delta x^2}(h_{i+1,j}^{t} - 2h_{i,j}^{t} + h_{i-1,j}) \\ &+ \frac{1}{\delta y^2}(h_{i,j+1}^{t} - 2h_{i,j}^{t} + h_{i,j-1}^{t})]\}, \end{aligned} \tag{11}$$

where the indices i, j, k refer to the cell in which the surface is located and $\bar{w}_{i,j,k}$ is the z component of the velocity vector at the surface and is calculated by linear interpolation between the w-velocity at the top and bottom faces of the surface cell.

5 Visualization of Splashing, and Fluid Configuration

Even though the Navier-Stokes equations are the physically correct model for fluid motion, additional visualization techniques need to be developed in cases where the fluid surface is not single-valued (e.g., splashing) or in order to visualize the fluid configuration (e.g., vorticity, flow). In those situations, we use a technique [4] which allows keeping track of fluid position. This technique supplies a coordinate system of marker particles whose trajectories follow the motions of elements *throughout* the fluid. It must be emphasized that the marker particles which we introduce into the incompressible fluid flow calculation are only for the purpose of indicating fluid configuration. They show which cells contain fluid and especially which cells lie along the free surface. They also serve as a flow visualization coordinate system whereby fluid element trajectories and relative positions can be observed.

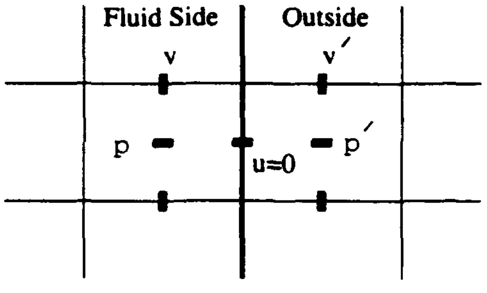


Figure 2: Free-Slip boundary conditions

In this technique, marker particles are initially placed in the cells containing fluid and they are subsequently moved with local velocity. A linear interpolation is performed to calculate the velocity with which a particle should move. The interpolation weighting depends upon the distance of the particle from the nearest velocity points in the mesh of cells. A cell with no parker particles is considered to contain no fluid. A cell with marker particles, lying adjacent to an empty cell, is called a surface cell. All other cells with particles are considered to be filled with fluid.

6 Boundaries

We distinguish two kinds of boundary conditions.

1. *Rigid Boundaries* associated with fully or partially submerged obstacles, enclosing rigid walls, inflow and outflow,
2. *Free Surface* boundary conditions.

For each boundary type we compute the boundary conditions as follows.

6.1 Rigid Boundary Conditions

We consider two kinds of rigid boundary conditions which are applied at the walls of the computing finite-difference mesh.

1. *Rigid wall* boundaries include fully or partially submerged obstacles and enclosing rigid walls. A rigid wall may be either of two types, *no-slip* or *free-slip*.
2. *Inflow or outflow* boundaries.

We now list the above four types of boundary conditions and the associated velocity and pressure calculation equations for the case of a two-dimensional flow with velocities u and v along the x and y axes respectively. The above calculation of boundary conditions can be similarly derived for the three-dimensional case.

1. *Free-slip* boundaries. A free-slip boundary represents a plane of symmetry or a non-adhering surface that exerts no drag upon the fluid. For a free-slip boundary, normal velocity reverses while tangential velocity remains the same. Also the normal velocity component, u, vanishes at the wall

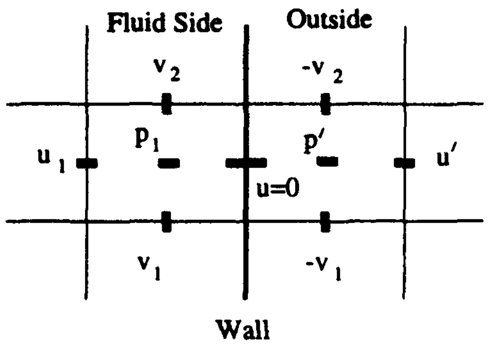


Figure 3: No-Slip boundary conditions

as shown in Fig. 2. Therefore, in the above diagram $v' = v$. Also the boundary condition for the pressure is $p' = p \pm g_x \delta x$. The sign is "+" if the fluid lies to the left of the wall and "−" if it lies to the right. For a horizontal wall $p' = p \pm g_y \delta y$, where the sign is "+" if the fluid lies below the wall and "−" if the fluid lies above.

2. *No-slip* boundaries. A no-slip boundary represents a viscous boundary that exerts a drag upon the fluid. For a no-slip boundary normal velocity remains the same, while tangential velocity reverses. For a vertical wall the pressure boundary conditions (see Fig. 3) are given by

$$p' \;=\; p_1 \pm g_x \delta x \pm (2\nu u_1/\delta x), \qquad (12)$$

where the sign is "+" for fluid to the left of the wall and "−" for fluid to the right of the wall. If the wall is horizontal then

$$p' \;=\; p_1 \pm g_y \delta y \pm (2\nu v_1/\delta y), \qquad (13)$$

where the sign is "+" for fluid below the wall and "−" for fluid above the wall.

3. *Inflow* boundaries. An inflow boundary allows the fluid to move into the system at a prescribed rate, i.e., $u = u_{prescribed}$ (see the above two figures), while the tangential velocity reverses.

4. *Outflow* boundaries. An outflow boundary allows fluid to pass out of the system at its own chosen rate, i.e., $\tilde{u} = \tilde{u}'$ (see the above two figures), while the tangential velocity remains the same.

6.2 Free Surface Boundary Conditions

Velocity boundary conditions at a free surface are based upon the requirement that $D = 0$ for surface cells. The easiest way to discuss is that of a two-dimensional surface cell which faces vacuum on only one side. Velocities for the other three sides are calculated in the usual manner, and that of the fourth side follows uniquely from the vanishing of D. Surface cells with two sides towards the vacuum are treated somewhat differently. For them we require $\partial u/\partial x$ and $\partial v/\partial y$ to both vanish separately; that is each vacuum-side velocity is set equal to the velocity on the side of the cell across from it. This also satisfies $D = 0$. A cell with three sides facing vacuum is relatively rare. The vacuum side opposite the fluid side is made to carry the velocity of that side; the other two vacuum sides which oppose each other are calculated to follow freely the effects of the body force and do not otherwise change. A surface cell with four sides towards the vacuum is similarly treated so that this isolated drop follows a free-fall trajectory. Finally, the pressure boundary condition at the free surface is computed by equating it to the applied external pressure.

7 Buoyancy

Given that at every iteration we have a complete set of velocities and pressures, we can use them to simulate floating dynamic objects. In particular, we assume that each rigid object is discretized and consists of a set of nodes n_i. For each model surface node n_i which is within the fluid, we compute the force acting on this node based on the following formula [4]

$$\mathbf{f}_{n_i} = -\nabla p_i dV_i + m_i \mathbf{g}, \qquad (14)$$

where dV_i is a volume associated with the submerged node of the object and ∇p_i is the gradient vector of the pressure and each component of it is computed in discrete form by the following formula

$$(\nabla p_i)_j = \frac{p_{n_i} - p_{n_i,j}}{\delta j}, \qquad j = x, y, z\ , \qquad (15)$$

where p_{n_i} is the pressure of the cell where the location of n_i is and $p_{n_i,j}$ is the location of the previous cell from the one where n_i is in the j direction. Also, $\mathbf{g}$ is the gravitational acceleration and m_i is the nodal mass assuming lumped masses. The total force on the object due to the fluid motion and gravity is given in discrete form by

$$\mathbf{f}_{fluid} = \sum_i \mathbf{f}_{n_i}. \qquad (16)$$

Based on the total force acting on each node, we compute the generalized external forces $\mathbf{f}_q$ (total force and torque acting on the object) as demonstrated in [12] and we compute its motion based on the Lagrange equations of motion

$$\mathbf{M}\ddot{q} + \mathbf{D}\dot{q} = \mathbf{f}_q + \mathbf{g}_q, \qquad (17)$$

where $\mathbf{M}$ and $\mathbf{D}$ are the object's generalized mass and damping matrices, and $\mathbf{g}_q$ are the generalized coriolis and centrifugal forces.

In order to handle collisions of the floating objects with static obstacles, we also apply the techniques developed in [12] for collision detection and collision force computation.

The floating objects that we use in our examples are small compared to the mesh size and therefore we make the simplifying assumption that they do not affect the water flow seriously. Otherwise, more sophisticated techniques need to be employed.

8 Numerical Stability Conditions

The first condition that is necessary to be satisfied to achieve numerical stability is that the distance the fluid travels in one time increment must be less than one space increment which results in

$$\delta t < min(\frac{\delta x}{|u|}, \frac{\delta y}{|v|}, \frac{\delta z}{|w|}). \tag{18}$$

Two additional kinds of instability which are related to the occurrence of truncation errors and have the form of "negative" diffusion require choosing a value of viscosity large enough to satisfy the condition

$$\nu > max[(\frac{\delta t}{2}u^2 + \frac{\delta x^2}{2}\frac{\partial u}{\partial x}), (\frac{\delta t}{2}v^2 + \frac{\delta y^2}{2}\frac{\partial v}{\partial y}), (\frac{\delta t}{2}w^2 + \frac{\delta z^2}{2}\frac{\partial w}{\partial z})]. \tag{19}$$

Finally, in case of viscous fluid simulations (i.e., ν is large) the condition necessary to insure stability through successive time steps is

$$\nu\delta t < \frac{1}{2}(\frac{1}{\delta x^2} + \frac{1}{\delta y^2} + \frac{1}{\delta z^2}). \tag{20}$$

9 Algorithm

In summary our algorithm for the solution of the Navier-Stokes equations including the boundary conditions can be summarized by the following steps if marker particles are used

- *Algorithm*
  1. Examine the location of the marker particles and identify the surface cells.
  2. Set-up the boundary conditions.
  3. Compute $\tilde{u}, \tilde{v}, \tilde{w}$ for all the cells which contain fluid.
  4. Perform the pressure iteration and update the fluid velocities u, v, w.
  5. Move the marker particles based on the new fluid velocities.
  6. Go to step 1.

If instead of marker particles we use the free surface equation (11), step 5 is replaced by the computation of the height of the free surface and in step 1 we identify the surface cells based on the computed height field.

10 Examples

We have created several two and three dimensional dynamic fluid animations to demonstrate the power of our technique. The animations involve phenomena like splashing, vorticity, waterfalls, waves, flow over fully or partially submerged objects and buoyancy in a fluid flow with rigid objects interacting with static obstacles[1]. We use standard SGI GL library routines to render water. We alter the transparency of the water based on the viewing angle and the surface normal if we have a height filed. Otherwise, we just draw particles for visualization of splashing and vorticity.

We also develop an editing tool for easy definition by a user of a physical-world which includes obstacles, boundaries and fluid properties such as viscosity, initial velocity and pressure. Using our editor we can then perform complex fluid simulations without knowledge of the underlying fluid dynamic equations. The first animation in the videotape shows the use of the editor to construct the physical environment for the animation shown in Figure 5 including the definition of boundaries and fluid properties such as viscosity.

Figure 4 shows several frames from a 2D animation of a splashing drop into a pool of *milk-like* liquid. The walls enclosing the liquid are free-slip, the grid size is 40 x 60 and the calculation time including the graphics took 20 minutes. Throughout the animation, marker particles are used to visualize the splashing. Figure 4(a) shows the initial configuration with the drop above the liquid. Figure 4(b) shows a frame from the initial splash of the drop in the liquid creating a crater, while Figure 4(c) shows a frame from the liquid rebounding to cover the crater and creating an upward indentation.

Figure 5 shows several frames from a 2D animation of a water jet splashing into a concrete tank. The walls of the tank are no-slip, the grid size is 40×60 and the calculation time including the graphics took 15 minutes. The water flows from an *inflow* type boundary marked in red. Throughout the animation, marker particles are used to visualize the splashing. This simulation took less time to compute than the previous, which is simpler, due to the fact that we used a smaller number of particles. By comparing images from both animations the marker density in the second animation is obviously smaller. Figure 5(a) shows a frame from the water jet before it hits the bottom of the tank. Figure 5(b) shows a frame from the initial splash of the water jet to the bottom of the tank and its subsequent splash on the vertical walls of the tank. Due to this second splash rotational stresses begin to develop around the vertical walls which result in the development of two vortices shown in Figure 5(c). Finally, the flow slows-down as more water pours into the tank and the vortices are not so prominent as shown in Figure 5(d).

Figure 6 shows several frames from a 3D animation of a forced wave traveling along a river bed with submerged and semi-submerged obstacles. The walls of the obstacles are free-slip, the grid size is 40 x 12 x 40 and the calculation time including the graphics took approximately one hour. The water surface height is calculated using the height field equation (11) which uses velocities computed from the solution of the Navier-Stokes equations. The forced-wave travels along the river bed due to the opening of a sluice gate whose other side contains water at a higher elevation. All the sides of the river bed do not allow water to outflow, causing the elevation of the water surface level and multiple reflections due to the water wave. Figure 6(a) shows the initial state where the various obstacles are clearly seen. Figure 6(b) shows the reflection by 90 degrees of the forced-wave due to the vertical wall of the river banks, while Figures 6(c-d)

[1] Note to reviewers: The examples we present here are also included in the accompanying videotape

show two frames from the wave breaking on the semi-submerged obstacle. Notice that the water level has been increased due to the inflow of water.

Figure 7 shows several frames from a 3D animation of a double non-forced deep ocean wavefront being reflected and diffracted off a harbor wall. The walls of the harbor are free-slip, the grid size is $50 \times 20 \times 40$ and the calculation time including the graphics took approximately 2 hours. The water surface height is calculated using the height field equation (11). Figure 7(a) shows a frame from the animation where the first wave has hit the harbor wall. Figure 7(b) shows the reflection of the first wave off the harbor wall, while Figure 7(c) shows the superposition of the first reflected frame with the second wave. Finally, Figure 7(d) shows the diffraction of the waves off the end of the harbor wall.

Figure 8 shows two frames from a 3D animation of a fast flow past a series of submerged and semi-submerged piles which includes buoyancy of two cylinders interacting with the piles. The grid size is $30 \times 15 \times 30$ and the calculation time including the graphics took approximately 33 minutes. The water surface height is calculated using the height field equation (11). Figure 8(a) shows a frame from the animation where the inflow has just started making the cylinders move as they float. Six different piles are clearly seen. Figure 8(b) shows a frame from the animation after the cylinder to the right has collided with the piles. The water around some of the piles is starting to rise as a result of the increased water pressure. Furthermore, water has completely covered one of the piles.

11 Conclusion

We have developed a new general approach for the modeling and animation of incompressible fluids. We used the full time-dependent Navier-Stokes equations to comprehensively simulate 2D and 3D incompressible fluid phenomena. Due to the increased demand for realism in present day animations, our first goal was to accurately model phenomena like shallow or deep fluid flow, transient dynamic flow, obstacles and waves that previous techniques could only approximately animate. Second, we simulated phenomena like vorticity and splashing which could not be modeled previously. Furthermore, we were able to model interactions of fluids with rigid floating objects by using the pressures and velocities computed by the solution of the Navier-Stokes equations.

Our numerical analysis technique for solving the Navier-Stokes equations was based on finite-differences approximations of those equations. To model free-surface fluids, we presented a powerful technique which is based on the Marker-and-Cell method. Based on the fluid's pressure and velocities obtained from the solution of the Navier-Stokes equations this technique allows modeling of the fluid's free surface either by solving a surface equation or by tracking the motion of marker particles. The later technique is suitable for splashing effects.

We developed an editing tool for easy definition of a physical-world which includes obstacles, boundaries and fluid properties such as viscosity, initial velocity and pressure. The editor allows the simulation of complex fluid flows without prior knowledge of the underlying fluid dynamic equations.

Finally, we demonstrated the power of our technique through a series of examples which required up to two hours of computation on an SGI Crimson workstation.

References

[1] Baraff, D., (1989) "Analytical methods for dynamic simulation of nonpenetrating rigid bodies," *Computer Graphics* (SIGGRAPH Proceedings), 23, pp. 223-232.

[2] Barzel, R., and Barr, A., (1988) "A modeling system based on dynamic constraints," *Computer Graphics*, **22**(4), 179-188.

[3] Chorin, A.J., (1968) "Numerical Solution of the Navier-Stokes Equations", Math. Comp., 22, pp. 745-762.

[4] Fletcher, C.A.J., (1990) "Computational Techniques for Fluid Dynamics," Springer Verlag, Sydney, 1990.

[5] Fournier, A. and Reeves, W., (1986) "A Simple Model of Ocean Waves," SIGGRAPH Proceedings, pp. 75-84.

[6] Harlow, F.H., and Welch, J.E., (1965) "Numerical Calculation of Time-Dependent Viscous Incompressible Flow," Phys. Fluids, 8, pp. 2182-2189.

[7] Kallinderis, Y., and Baron, J., (1989) "Adaptation methods for a new Navier-Stokes algorithm," AIAA Journal, 27(1), pp. 37-43.

[8] Kass, M., and Miller, G., (1990) "Rapid, stable fluid dynamics for computer graphics," *Computer Graphics*, **24**(4), 49–57, (Proc. SIGGRAPH).

[9] Lewis, J., (1987) "Generalized Stochastic Subdivision," ACM Transactions on Graphics, 6(3), pp. 167-190.

[10] Max, N., (1981) "Vectorized procedural models for natural terrain: Waves and islands in the sunset," SIGGRAPH Proceedings, pp. 317-324.

[11] Masten, G., and Wartterberg, P., and Mareda, L., (1987) "Fourier Synthesis of Ocean Scenes," IEEE Computer Graphics and Application, 7(3), pp. 16-23.

[12] Metaxas, D., and Terzopoulos, D., (1992) "Dynamic Deformation of Solid Primitives with Constraints," SIGGRAPH Proceedings, pp. 309–312.

[13] Miller, G., and Pearce, A., (1989) "Globular Dynamics: A connected particle system for animating viscous fluids," Computer Graphics 13(3), pp. 305-309.

[14] Miyata, H., and Nishimura, S., (1985) "Finite difference simulation of nonlinear waves generated by ships of arbitrary three-dimensional configuration," Journal of Computational Physics, 60, pp. 391-436.

[15] Peachy, D., (1986) "Modeling Waves and Surf," SIGGRAPH Proceedings, pp. 65-74.

[16] Perlin, K., (1985) "An Image Synthesizer." SIGGRAPH Proceedings, pp. 287-296.

[17] Schachter, B., (1980) "Long crested wave models," Computer Graphics and Image Processing, 12, February, pp. 187-201.

[18] Shabana, A., (1989) *Dynamics of multibody systems*, Wiley, New York.

[19] Ts'o, P., and Barsky, B., (1987) "Modeling and rendering waves," ACM Transactions on Graphics, 6(3), pp. 191-214.

[20] Wittenburg, J., (1977) *Dynamics of systems of rigid bodies*, Tubner, Stuttgart.

[21] Zhang, R., and Mustoe, G.G.W., and Nelson, K.R., (1993) "Simulation of Hydraulic Phenomena Using the Discrete Element Method," Proc. of the 2nd International Conference on Discrete Element Methods, MIT, pp. 189-200.

(a) (b) (c)

Figure 4: Splashing drop. Initial state (a). Splashing and rebounding (b,c).

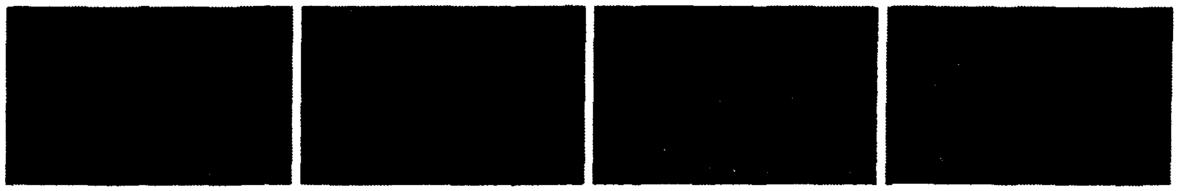

(a) (b) (c) (d)

Figure 5: Water jet splashing into a concrete tank (a-d).

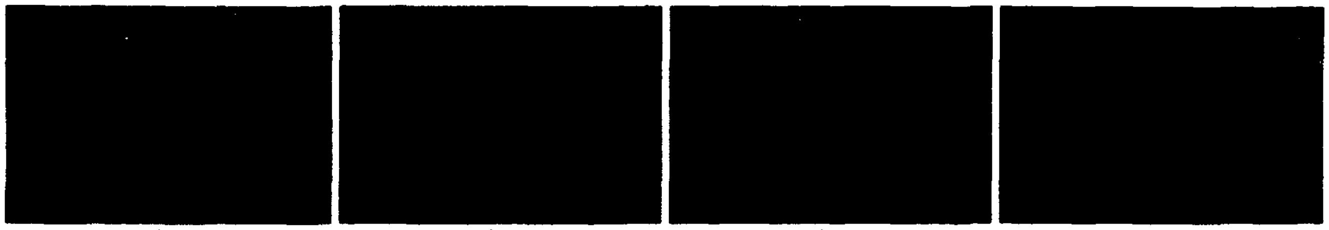

(a) (b) (c) (d)

Figure 6: Forced-wave traveling along a river bed with submerged and semi-submerged obstacles. Initial state and obstacles (a). Wave reflection by 90 degrees due to an obstacle (b). Wave breaking on a semi-submerged obstacle (c,d).

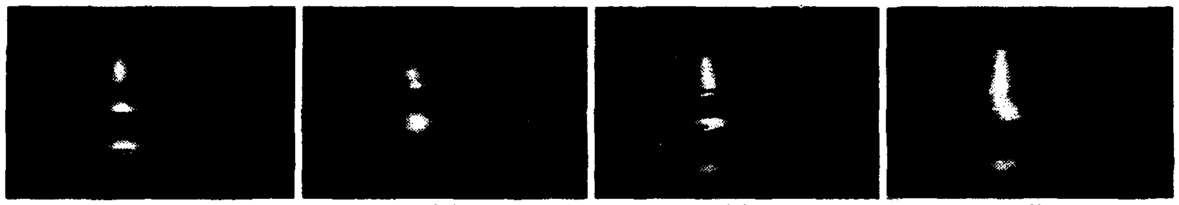

(a) (b) (c) (d)

Figure 7: Double wavefront reflected and diffracted off a harbour wall. First wave (a). Reflection of first wave off harbour wall (b). Superposition of the first reflected wave and second wave (c). Wave Diffraction (d).

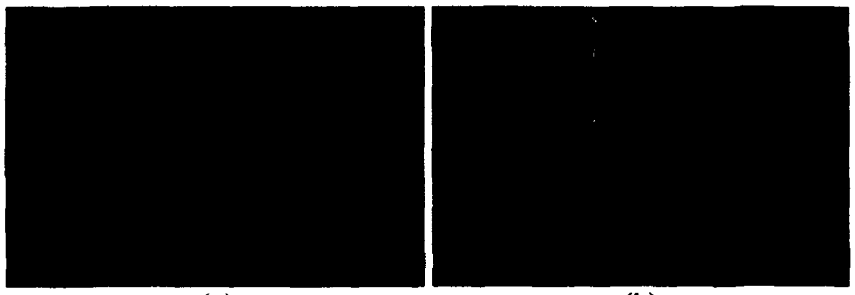

(a) (b)

Figure 8: Fast steady flow and buoyancy around obstacles. Initial state (a). State after collision of the cylinder to the right with a pile (b).

H Animation and Control of Four-Legged Animals: Evangelos Kokkevis, Dimitri Metaxas, and Norm Badler

Animation and Control of Four-Legged Animals

Evangelos Kokkevis, Dimitri Metaxas and Norm Badler
Department of Computer and Information Science
University of Pennsylvania, Philadelphia, PA 19104

Abstract

This paper develops a new approach to the animation and motion control of four-legged animals. Even though our approach is general we use it to animate realistic walking and trotting of four-legged animals, as well as low level behaviors like target following. We first use model-based control algorithms to control the velocity, position and posture of the animal's body. The output of the dynamic controller is a set of forces and torques that are then distributed to the legs of the animal based on linear programming. As a result of those forces and torques the ground exerts forces that make the animal move. In order to ensure realistic walking and trotting animations at any given animal speed, animal posture and ground elevation we then develop a gait controller that controls the motion pattern of the legs. The gait controller based on the fluctuating speed and orientation of the animal body computed by the dynamic controller, kinematically determines the sequence in which legs are lifted off the ground, their speed when they are off the ground and their position when they retouch the ground. This mixture of control theory and kinematic gait control creates more realistic animations compared to previous approaches to the same problem which used purely kinematic solutions, or only control theory or open-loop control and kinematics. Furthermore, based on the forces computed by the controller and the knowledge of the ground's frictional force we can automatically detect whether a desired given speed can be achieved based on the ground conditions and the joint torque limits of the animal. The low level behaviors that our algorithm models such as speed, posture, gait, gait transitions and target following can be used by a higher level system that controls more complex animations. We demonstrate our technique through a series of dog animations.

Category: Research paper, **Format:** Regular paper.

Keywords: Control of Multi-legged Animals, Control theory and Kinematics, Control of low level behaviors, Physics-Based Modeling.

0

1 Introduction

A very important goal of computer graphics has been the realistic animation and control of living organisms including their behaviors. The major challenges which arise in such animations are first the complexity of modeling such organisms which include choosing the number of degrees of freedom, deciding which degrees of freedom are involved in a particular motion and modeling different motions. Second, is the problem of controlling the degrees of freedom to achieve the desired motion in a way that is both realistic and need only high level interaction of a user with the given system. Third, is the problem of intelligent behavior and interaction of the animal with its environment which includes low level behaviors like like obstacle avoidance and high level behaviors like intelligent decision making. This paper presents a new approach to the animation and control of multi-legged animals and low level behaviors that uses both model-based control theory and kinematic gait controllers.

Recently, several researchers have used a variety of techniques to address some of the problems within that general area. Girard [2, 1] develops a kinematic control solution to the problem of animating both human figures and multilegged animals. His method is based on rules which are associated with dynamics. Raibert and Hodgins [3] develop a control theory approach to model behaviors like running, trotting, galloping, hopping and running of legged robots and animal bipeds. Their technique is based on hopping control, speed control and posture control. Witkin and kass [4] using optimization techniques develop a spacetime constraint algorithm and demonstrate its power by controlling a dynamic lamp. Panne, Fiume and Vranecic [5, 6] use optimal control theory in various applications like parking a car and for the dynamic walk of a biped across a variable terrain. Wilhelms et al [7, 8] develop techniques for controlling dynamic systems based on position control implemented with springs and dampers and other behaviors such as attraction and avoidance. Bruderlin and Calvert [9] develop a dynamic model that is kinematically controlled based on techniques adapted from biomechanics. Once the walking motion was calculated the rest of the body motion was base on kinematics. Finally, McKenna and Zeltzer [10] develop an open-loop dynamic system for a cockroach that is controlled by a kinematic gait controller which in turn triggers actuators to produce the necessary forces for various walking patterns.

In this paper we use both model-based control theory and kinematic gait controllers to achieve realistic animal animations and low level behaviors such as target following and reaching. It is worth mentioning though that our approach is general and can be applied with appropriate modifications to biped locomotion as well. Our approach is closest in nature to that used by McKenna and Zeltzer with the main difference being the use of closed-loop model-based control. In this way we can avoid the problems of manually tuning the action of motor programs to exert the correct forces that achieve the desired posture for example. Furthermore, we can simulate apart from walking other behaviors like animal trotting and furthermore target following at variable speeds determined by the target's motion.

We first use model-based control algorithms to control the velocity, position and posture of the animal's body. The output of the controller is a set of forces and torques that are then distributed to the legs of the animal based on linear programming. As a result of those forces and torques the ground exerts frictional and vertical forces that result in forces and torques that make the animal move.

In order to ensure realistic walking and trotting animations at variable animal speeds, animal posture and ground elevation we develop a kinematic gait controller that controls the motion pattern of the legs. The gait controller based on the fluctuating speed and orientation of the animal body computed by the dynamic controller, kinematically determines the sequence in which legs are lifted off the ground, their speed when they are off the ground and their position when they land on the ground. The use of control theory in combination with the kinematic gait controller creates very realistic animations. While with dynamic control theory we can control the speed of the animal and its posture, we can't control for example, the motion of the legs once they are off the ground or the sequence in which the legs are lifted from the ground. A kinematic controller though can achieve that very easily.

Furthermore, based on the forces computed by the dynamic controller and the knowledge of the ground's frictional force we can automatically detect whether a desired given speed can be achieved based on the ground conditions and the joint torque limits of the animal. The low level behaviors such as speed, posture, gait, gait transitions and target following that our algorithm models can be used by a higher level system that controls more complex animations.

In the following sections we present the details of our technique whose dynamic control part is similar to Park's approach for controlling dynamic vehicles [11] and demonstrate it through examples involving dog animations.

2 The Animal Model

Although the control methods used in this paper are general and can be applied to a variety of legged animals with two or more feet, we decided to concentrate on four legged creatures. The model that we use to demonstrate our theory is that of a dog shown in Figure 1. The number of degrees of freedom (dof) of most legged animals is very large primarily due to the flexibility of their body and the complexity of their joints. This makes it almost impossible to accurately control any artificial model, unless certain simplifying assumptions are made. In our case, we assume that the animal's body is rigid. Several articulated parts are connected to the body with joints. These are the four legs, the neck with the head and the tail. The front legs consist of two joints: the hip joint that connects them to the body and the ankle joint that connects the upper and lower parts of the leg. The rear legs have an extra joint that connects the lower leg part to the leg's paw. The hip joints have two rotational and two translational dofs. The ankle and the paw joints have only one rotational dof. The neck and the tail are jointed to the body with a one dof rotational joint. Finally the head is connected to the neck with a two dof rotational joint.

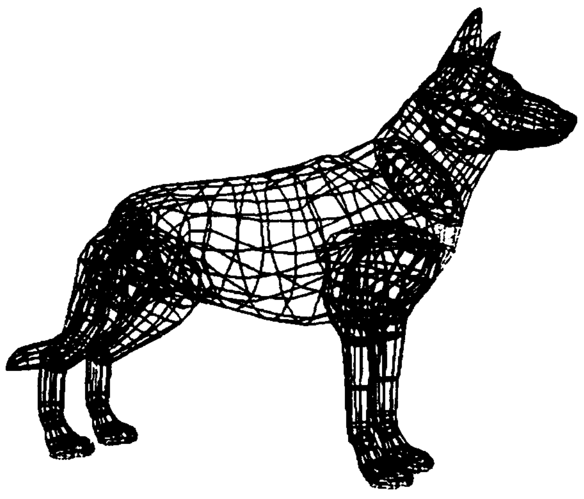


Figure 1: Dog model.

3 Model-Based Control

To be able to control a given system (passive or active) we need first to define it dynamically. In control theory terminology the dynamic model we use to model the dynamics of a given system is called the plant. Once the plant has been defined we apply some control strategy to be able to control its behavior of certain of its variables.

The way we model the problem of a walking or trotting animal is through the design of two subsystems. One subsystem is the rigid body of the animal, while the other subsystem includes everything else, i.e., its legs, the terrain etc. These two subsystems, the body subsystem and the leg subsystem, interact through forces. The forces exerted by the legs result in acceleration vectors on the body. At the same time the instantaneous position of the body determines which legs are able to touch the ground and exert a force. The velocity of the body determines how fast the leg joints have to flex to keep the feet on the ground. Thus the two subsystems influence one another in a closed-loop feedback network.

In this paper we adopt the following simplifying assumption for the animal's body. The body can be completely described by it mass m, its inertia tensor $\mathbf{I}_\theta$, its dimensions with respect to a coordinate frame ϕ located at its center its center of gravity, and the following quantities with respect to a world coordinate system Φ,

$$\mathbf{s} = \left\{ \begin{array}{c} \mathbf{c} \\ \dot{\mathbf{c}} \\ \theta \\ \omega \end{array} \right\}, \tag{1}$$

where $\mathbf{c}$ and $\dot{\mathbf{c}}$ are the position and velocity vectors of the body's center of gravity, θ is the orientation of the body (i.e., the rotation of ϕ w.r.t. Φ) expressed as Euler angles and ω is the angular velocity of the body. The above variables are the state variables of our body subsystem. Their initial values can be specified arbitrarily, but afterwards they will vary according to the accelerations applied by the legs. We also express the position of a point on the animal $\mathbf{x}$ w.r.t. to the world coordinate system Φ as

$$\mathbf{x} = \mathbf{c} + \mathbf{R}\mathbf{p}, \tag{2}$$

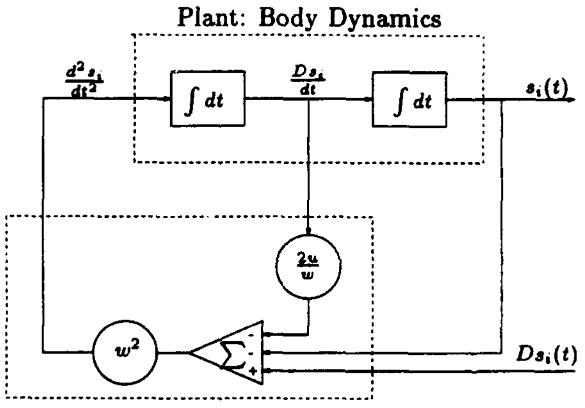
where $\mathbf{R}$ is the rotation matrix corresponding to θ and $\mathbf{p}$ is the position of the point w.r.t. to ϕ.

To control the vehicle position completely it is sufficient to control the vectors $\mathbf{c}$ and θ. In practice though is is often more desirable or convenient to control the rate of change of these vectors, i.e., to control the body's linear and angular velocity. The only way the controller can control the animal's motion is by exerting forces through the legs (which simulate the action of muscles). All the leg forces acting together result in a net force and a net torque at the center of gravity of the body. These in turn result in the linear and angular acceleration vectors $\ddot{\mathbf{c}}$ and $\dot{\omega}$ from which we update the animal's position in space. Therefore, the dynamic controller must control the body's state variables indirectly through those accelerations.

The design of the above controllers depends on certain minimum criteria that need to be met to ensure that the walking will look realistic. We came up with the following three minimum criteria.

1. The body of the animal should never hit the ground.
2. The animal should be able to stay on course even when moving on rough terrain.
3. The animal's leg motion should be regular on good terrain, but not necessarily on bad terrain.
4. We want to be able to control the animal's speed to be able to model low level behaviors like target tracking.

Based on the above criteria we choose to control the three orientation angles θ_i of the center of gravity (c.o.g.), the vertical position of the c.o.g. c_z and the two horizontal velocities $\dot{c}_x$ and $\dot{c}_y$ of the c.o.g. In this way we can control both the speed and the location of the animal. Without loss of generality, we can approximate the body subsystem as six noninteracting subsystems. Each of those subsystems represents the dynamics of the rigid body of the animal. The first three subsystems represent the way the vehicle body rotates about its center of gravity. The other three represent the way the center of gravity moves through space. Each of those subsystems has the following form. Given as input the acceleration of the corresponding variable resulting from the forces and torques the legs exert, each subsystem through one or two consecutive integrations (depending on whether it controls a velocity or position variable) computes the value of the corresponding parameter. In control theory terminology each of those subsystems is called a double-integral or single-integral plant (see Fig 2 for a double integral plant). And all six of them comprise the body's plant.



Controller

Figure 2: Double Integral plant and controller used for position variables.

The problem then of controlling the body motion reduces to the simpler problem of controlling the state variables of the six non-interacting double integral plants defined above. To control each of those plants we set-up a control loop in each of them to regulate one of its state variables as shown in Figure 2 for the plant which corresponds to a system variable s_i. Ds_i is the desired value of the parameter s_i. The block labeled s_i-controller will produce an acceleration $\ddot{s}_i$ in order to make s_i equal to Ds_i. The accelerations (linear or angular) exerted by the controllers are a result of forces and torques that all the legs should simultaneously exert as a result of muscle action. The question that arises is whether such forces and torques can be exerted by the legs. Depending on the animal's torque joint limits and the friction that the ground can exert it may not be possible. In such a case we will assume that the animal's legs will exert the maximum force or torque they can. This in turn means that the desired value of the controlled variable will take longer or may not be possible to be reached. We will always assume for the purposes of this paper that the animal exerts the maximum force it can to avoid slipping or damaging its joints.

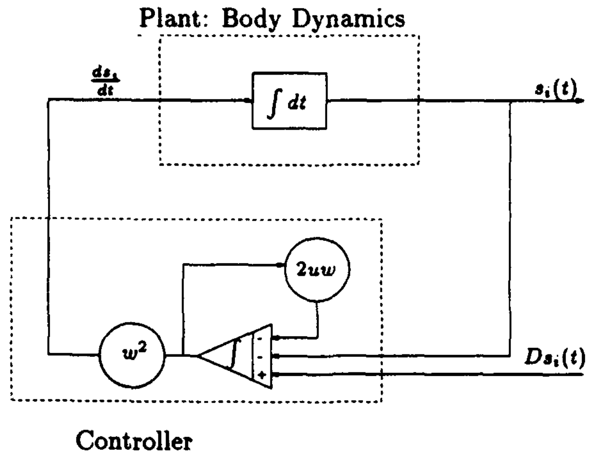


Figure 3: Single integral plant and controller used for position variables

We chose the control laws by which the controllers were designed so that the closed-loop response of each control loop would be of the general form expressed in the complex frequency domain (after taking the Laplace transform)

$$\frac{s_i(s)}{Ds_i(s)} = \frac{1.0}{\frac{s^2}{w^2} + \frac{2us}{w} + 1.0}. \quad (3)$$

In this expression, u is the damping ratio of an ideal control loop, w is its undamped natural frequency and s is the complex frequency. The Laplace transform of the controlled variable is $s_i(s)$, while the Laplace transform of the time-varying desired value of that variable is $Ds_i(s)$. The u and w parameters specify the damping and speed of response of the controller to a desired value and are set by the user.

In our system, controllers which regulated a position variable controlled the output of a double-integral plant. The other controllers which regulated the horizontal velocity variables controlled single-integral plants. We therefore designed two controllers which had the same closed-loop response (3). The controllers for the double-integral plants contained two gain blocks and a summation block as shown in Figure 2. The controllers for the single-integral plants contained a lag network followed by gain, as shown in Figure 3.

4 Linear Programming

As mentioned above the dynamic controllers compute the total force and torque that needs to be applied to the center of mass of the animal body, so that the body obtains the desired velocity, position and orientation. This force and torque should be the result of forces exerted by the ground through the legs. We therefore need an algorithm that determines how the total force and torque should be distributed to the legs. The fraction of the total force that each leg will exert to the ground is determined using linear programming. Clearly, only the feet that are in contact with the ground each time are capable of moving the body. The orientation of the body is controlled by the torques applied at the center of mass. These torques result from the force applied by each leg and depend on the relative position of the tip of that leg with respect to the center of the body. Limits exist on the maximum force that each leg can exert. This means that, depending on the number of legs in contact with the ground and their relative position, it might or might not be possible to apply the required forces and torques (computed by the dynamic controllers). The purpose of the linear programming procedure is to find the optimal force assignment to legs which minimizes the difference between the required and the actual values.

Linear programming was chosen for the force distribution for its versatility and efficiency. The Simplex method that is most often used to solve linear programming problems can be expected to run in linear time for most of the cases [12]. Linear programming attempts to assign values to variables so that a specific cost function, depending on these variables, is minimized subject to certain constraints. In our case the variables are the actual forces exerted by each leg. The constraints are the limits in the maximum force that each leg can exert, and the cost function consists a weighted sum of the differences between required and actual forces and torques. Ideally we would like the actual forces and torques to be equal to the required ones. In this case all the errors are zero, and the cost function is minimized. Adjusting the weights in the cost function is equivalent to changing the relative importance of matching any one of the variables with its required value. For example, if the cost for the error in the force on the x-axis is larger than then cost for the error in the torque around the y-axis, then the actual force in the x-direction will try to match as closely as possible its required value, even if this results to a bigger error in the y-axis torque. An extra term in the cost function is also added to enforce even distribution of the vertical force (mostly due to the body's weight) to all legs. This proves to add in the stability of the animal, especially in the case when one of the legs unexpectedly loses contact with the ground.

Linear programming turns out to perform very well in cases where the errors are small. However, when the required and actual forces and torques cannot be matched sufficiently closely, then even small changes in the weights of the cost function can give highly unbalanced errors.

5 Gait Controller

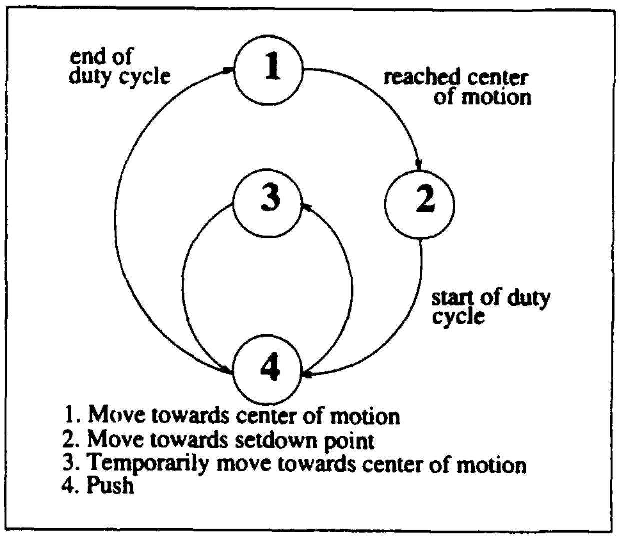
Gaits of different animals have been studied extensively and lots of interesting observations can be found in zoology literature [13, 14, 15, 16, 17]. Since our purpose is to blend kinematic motion together with dynamics, very valuable sources have been proven to be [17] and [16], in which snapshots of animals walking and trotting are presented. Some technical terms needed for the description of gaits can be found in [15] and we repeat them here. A *stride* is a complete cycle of leg movements, for example the sequence from the setting down of a particular foot to the next setting down of the same foot. A *stride length* is the distance traveled in a stride and *Stride frequency* is the number of strides taken per unit time. The *duty factor* of a foot is the duration of the stride for which it is on the ground.

The main purpose of the gait controller is to reposition the legs depending on the linear velocity and turning rate of the animal's body. We have implemented two different gaits, the *walk* and the *trot*. The dog automatically switches between them, depending on its speed. The walking gait is used by dogs when they move slowly. When walking on flat surfaces with constant velocity, three legs are always in contact with the ground and one is lifted. The order in which legs a lifted during a stride is: LF, RH, RF, LH (where L, R, F, H are for Left, Right, Front, Hind, respectively). This order ensures that the center of mass is always within the triangle formed by the tips of the legs on the ground, guaranteeing static stability for the animal. During the walk, the duty factor for each foot is 0.75. At faster speeds, dogs use the trot. During the trot, LF and RH move together, so do RF and LH. The duty factor for each foot is 0.5 and at any time two feet are on the ground and two feet are lifted. During the trot the dog has to achieve dynamic stability.

The gait controller is responsible for the kinematics of the leg motion. When a leg is touching down, its motion is guided by the ground reaction forces. This enhances the realism of the animation. However, when the leg is lifted from the ground, there is no reason to keep on treating it dynamically since its motion has negligible effects on the motion of the rest of the body. The fact that a particular animal if trained appropriately, can walk in a variety of styles supports even more the claim that dynamics alone are not sufficient to create realistically looking motion.

The inputs to the gait controller are the desired speed of the animal as well as its angular velocity. The higher the speed, the longer the stride length and the stride frequency. This means that the swing angle of a leg while lifted from the ground increases with an increase in the animal's speed. When the animal is turning, the gait controller positions the legs not only forward but also sideways so that they can apply the appropriate torque for the turn to take place.

The leg motion is guided by the automaton shown in Figure 4. Each leg can be in one of four distinct states. Actions that cause a change in the leg's state are triggered either externally by the gait controller or internally as a result of the leg's orientation. State 4 is the only state where a leg is touching the ground and pushing. When the duty interval of the leg is over, the gait controller triggers a transition from state 4 to state 1 for that leg. In state 1 the leg moves towards the center of its swing (center of motion). When it reaches



end of duty cycle
1
reached center of motion
2
3
start of duty cycle
4
1. Move towards center of motion
2. Move towards setdown point
3. Temporarily move towards center of motion
4. Push

Figure 4: Automaton for leg motion transitions.

that point it starts moving towards it's *set down point*. The set down point is determined by the gait controller, and depends on the linear and angular velocity of the body, as explained above. Finally, when it's time for the duty interval to begin, the gait controller once more triggers a transition, this time from state 2 to state 4. The leg goes into state 3, if it gets jammed. Legs get jammed as a result of unexpected events that cause the hip joint to rotate outside the operational area of the leg. Sliperry ground or very rough terrain could cause a leg to jam. Since the leg has to get back into its operational area, it gets lifted temporarily and moves for a short period of time towards the center of motion, until it reaches a safe position where it can go back to state 4. It is desirable for a leg to stay for short periods of time in state 3, since its being unexpectedly lifted creates potential hazards to the stability of the body.

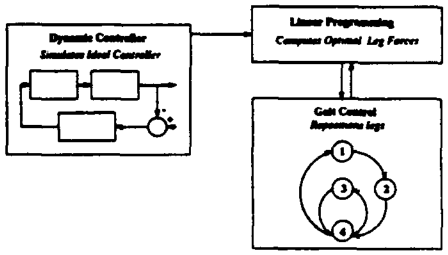


Figure 5: Flow diagram of our system

While a leg is in states 1, 2 or 3, the gait controller is responsible for kinematically setting the angles of its ankle and paw joints so that their motion looks as natural as possible. It turns out that the correct motion of these joints adds a lot to the visual effect of the animation.

In conclusion our approach consists of the dynamic controller, the linear programming and the gait controller subsystems that interact as shown in Figure 5.

6 Experiments

We have created two dynamic animations to demonstrate the power of our technique. The animations involve walking, trotting, transitions between these kinds of locomotion, turning, and acceleration and deceleration as part of simple behaviors like variable speed target pursuit. The examples we present here are included in the accompanying videotape.

The first animation shows several frames from a 3D animation of a dog going after a moving ball. In the first part of the animation the ball is initially static. The dog starts walking towards the ball and gradually changes its locomotion to trotting. Once the dog reaches the ball it stops, but the ball moves again in the second part of the animation. The dog slows its trotting speed and then increases its speed while simultaneously turning, to reach the ball Throughout the simulation the dynamic controller controls the speed, turning angle and height of the dogs body from the ground.

The second animation involves walking and trotting on uneven terrain. The dog starts from a still position and walks while turning towards the first bump. It then gradually starts to trot and then ascends and descends from the first bump. The dog then continues to move towards the dog-house and ascends on the second bump and then stops. Throughout the simulation the dynamic controller controls the speed, turning angle and height of the dog's body from the ground.

7 Conclusions

In this paper we have developed a new approach to the animation and control of four-legged animals by combining dynamic control theory algorithms and kinematic gait controllers. The dynamic controller regulates the position and velocity of the animal's body center of mass and computes the necessary forces and torques that are distributed to the animal's legs using linear programming. Furthermore, we developed a kinematic gait controller to control the motion pattern of the legs once they are off the ground. In this way we were able to animate realistic walking, trotting, transitions between these kinds of locomotion, turning, and acceleration and deceleration as part of simple behaviors like variable speed target pursuit.

References

[1] Girard, M., and Maciejewski, A., (1985) "Computational Modeling for the computer animation of legged figures", SIGGRAPH Proc., pp. 263-270.

[2] Girard, M., (1987) "Interactive design of 3D computer animated legged animal motion", IEEE Computer Graphics and Animation, pp. 39-51.

[3] Raibert, M., and Hodgins, J., (1991) "Animation of dynamic legged locomotion", SIGGRAPH Proc., pp. 349-358.

[4] Witkin, A., Kass, M. (1988) "Spacetime Constraints", SIGGRAPH Proc., pp. 159-168.

[5] van de Panne M., Fiume, E., Vranesic, Z., (1990) "Reusable Motion Synthesis Using State-Space Controllers", SIGGRAPH Proc., pp. 225-234.

[6] van de Panne M., Fiume, E., Vranesic, Z., (1993) "A controller for the dynamic walk of a biped across variable terrain", Proc. of the 31st IEEE conference on decision and control.

[7] Wilhelms, J., (1986) "Virya-A motion control editor for the kinematic and dynamic animation," Graphics Interface, pp. 141-146.

[8] Wilhelms, J. and Skinner, R., (1989) "An interactive approach to behavioral control", Proc. of Graphics Interface.

[9] Bruderlin, A., and Calvert T.W., (1989) "Goal-directed dynamic animation of human walking", SIGGRAPH Proc., pp. 233-242.

[10] McKenna M., and Zeltzer, D., (1990) "Dynamic Simulation of autonomous legged locomotion", SIGGRAPH Proc., pp. 29-38.

[11] Park, W. T., (1972) "Control of multilegged vehicles", PhD thesis Univ. of Pennsylvania.

[12] Press, W., Flannery, B., Teukolsky, S., and Vetterling, W., (1986) "Numerical Recipes: The art of scientific computing", Cambridge Univ. Press, Cambridge.

[13] Goldspink, D., Alexander, R Mc N, editors, "Mechanics and energetics of animal locomotion."

[14] Sukhanov V.B., (1968) "General System of symmetrical locomotion of terrestrial vertebrates and some features of movement of lower tetrapods, Nauka Publishers.

[15] Alexander, R.McN., (1984) "The gaits of bipedal and quadrupedal animals," The International journal of Robotics research, 3(2).

[16] Brown M.C., (1986) "Dog Locomotion and gait analysis".

[17] Muybridge E., (1957) "Animals in Motion," Dover Publications, NY 1957.

I Pipeline Rendering: Interactive Refractions, Reflections, and Shadows: Paul J. Diefenbach and Norman I. Badler

Pipeline Rendering:Interactive Refractions, Reflections, and Shadows

Research Paper: Regular Format

Paul J. Diefenbach and Norman I. Badler

Department of Computer Science, University of Pennsylvania, Philadelphia PA 19104

Contact:Paul J. Diefenbach
Computer Graphics Research Lab
Moore School
University of Pennsylvania
Philadelphia PA 19104

Phone: (215) 898-1976
Fax: (215) 898-0587
Email: diefenba@graphics.cis.upenn.edu

Pipeline Rendering:Interactive Refractions, Reflections, and Shadows

Research Paper: Regular Format

Abstract

A coordinated use of hardware-provided bitplanes and rendering pipelines can create ray-trace quality illumination effects in real-time. We provide recursive reflections through the use of secondary viewpoints, and present a method for using a homogeneous 2-D projective image mapping to extend this method for refractive surfaces. We extend the traditional use of shadow volumes to provide reflected and refracted shadows as well as reflected and refracted lighting. A shadow blending technique is demonstrated, and the shadow and lighting effects are incorporated into our recursive viewpoint paradigm. Finally, we incorporate material properties including a translucency model to provide a general framework for creating physically approximate renderings. These techniques are immediately applicable to areas such as 3D modeling, animation, and interactive environments to produce more realistic images in real-time.

Keywords: *Real-time, Rendering Pipeline, Reflection, Refraction, Translucency, Shadows, Animation*

1 Introduction

Much attention has been devoted to photo-realistic rendering techniques as ray-tracing and radiosity packages have become increasingly sophisticated. These methods provide a basic foundation of visual cues and effects to produce extremely high quality and highly accurate images at a considerate cost, namely, computation time. Neither of these techniques have any widespread application in true interactive environments, such as animation creation and virtual worlds.

Many so-called interactive environments such as Virtual Building systems [1][17] rely on precomputation of static environments to form progressive radiosity solutions. Other systems dealing with lighting effects [7] rely on a series of images from a single viewpoint. All of the systems suffer from large computational overhead and unchangeable geometry. Even in incremental radiosity solutions [6], geometry changes require significant recomputation time. In addition, radiosity-based solutions inhibit the use of reflective and refractive surfaces.

Systems based on forward ray-tracing [8] are either non-interactive or else suffer from the problems inherent in the technique [20]. Only a few attempt to accurately handle indirect illumination [12]. Backward ray-tracing systems [2][10][5] more accurately handle caustics; but again these methods are very time-intensive and not remotely interactive. Even the fastest ray-tracing systems require static geometry to achieve their results [16].

In contrast, advanced hardware architectures such as the SGI Reality Engine^{TM} have brought an added level of realism to interactive environments through the use of sophisticated graphics pipelines and added levels of screen buffer information. These features have enabled software developers to bring previously unavailable details such as shadows and mirrors to many interactive applications. Even the most basic graphics systems today now support some level of image masking and manipulation common to the image processing community for years. These hardware provisions are have yet to be fully exploited, though clever programming techniques by several implementors have produced real-time shadows and mirrors.[11][14]

This paper expands these techniques to include not only reflection but a technique for refractive surfaces as well. The model presented extends the current reflection techniques to provide an arbitrary level of refraction and reflection for use in "hall-of-mirror" type environments and to provide a close approximation for refractive objects. A corrective image transform is presented to correct for perspective distortions during the image mapping of the secondary refracted image. In addition, a method for combining the previously exclusionary shadow and mirror stenciling methods is demonstrated which not only preserves shadows in all secondary images, but which also accounts for refraction and reflection of the light and shadows in the primary and secondary images as well. Finally, the use of hardware provided features such as fog and texture blending is shown to provide simulation of varying material properties such as translucency and shininess. Combined, these techniques provide a real-time alternative to ray-tracing for creating fast, approximate reflective and refractive lighting effects. Furthermore, the techniques described provide a foundation for more advanced rendering features such as anisotropic reflections and caustics.

2 Definitions

For the purposes of this paper, we shall introduce terms common to users in the GL environment. *Stencil planes* are essentially an enhanced Z-buffer mentioned in [3]. In its simplest form, pixels are written only if the current stencil value (analogous to the current Z value) of the destination pixel passes the defined stencil test. Depending on the result, the pixel is written and the stencil value is changed.

Shadow volumes are volumes bounded by silhouette faces. A silhouette face is a face created for each edge of an object by extending that edge away from the light source along the light-ray direction.

[0]Note to reviewer: The color figures included in this paper are also included as color slides. The slides are of much better quality.

An *accumulation buffer* is a secondary image buffer to which the current image can be added. The resulting image can also be divided by a constant. This enables a blending of images or image features.

In-out refractions are refractions which occur when light passes from one medium to another and back to the first, such as light traversing through a piece of glass. There is an entry refraction and an exit refraction, producing a refracted ray parallel to the incident ray.

3 Reflections

Reflections are a useful tool in interactive modeling and an important element for creating realistic animations. A reflective images corresponds to an inverted image from a secondary viewpoint. In other words, the reflected image is the flipped image from a viewpoint on the "other" side of the mirror. This analogy provides the basis for mirror reflection in systems such as [14].

Mirrors are implemented by rendering the entire environment, exclusive of the mirrored surface. The mirrored surface is drawn with Z-buffering, creating a stencil mask on pixels where the mirror is visible. A second rendering of the environment is then performed from the reflected viewpoint, drawing only over the previously masked pixels. Because the reflected angle (angle from mirror plane to reflected viewpoint) is the negative of the incident angle and because the image is flipped, the reflected image directly "fits" onto the mirror.

4 Refractions

Just as reflections provide strong visual cues, refractive elements also add additional realism for animations and interactive enviornments. Refractive images are similar in concept to reflections, but more complex in practice.

While a mirrored image directly corresponds to the reflective surface to which it maps, a refracted image maps to a distorted image space. Simply performing a second rendering in the stencilled area does not overlay the correct image portion. This is demonstrated in Figure 1. The area visible through the transparent surface in the refracted view is different than the image area from the original viewpoint: areas outside the refracting surface and even in front may be visible in the refracted image. This difference is due to two factors: the difference between incident and refracted viewpoints and the perspective distortion.

Because the incident angle does not equal the refracted angle, the refracted image is rotated with respect to the original image. This is further compounded by the rotated image plane undergoing a perspective distortion different than the perspective distortion of the original plane. The perspective transformations are the same, but because the planes have different orientations, the resulting distortions are different. The result is that a refractive square planar face, for example, maps to two different quadrilaterals in the original versus the refracted images.

The refractive image $\mathbf{I_r}$ does correspond to the original image $\mathbf{I_o}$ through a 2-D bijective projective mapping $\mathbf{M_3}$. This mapping is the intersection of the 3-D image mapping set $\mathbf{M_4}$ with the reflective planar surface Ψ:

$$\mathbf{I_o} = \mathbf{I_r M_3} \tag{1}$$

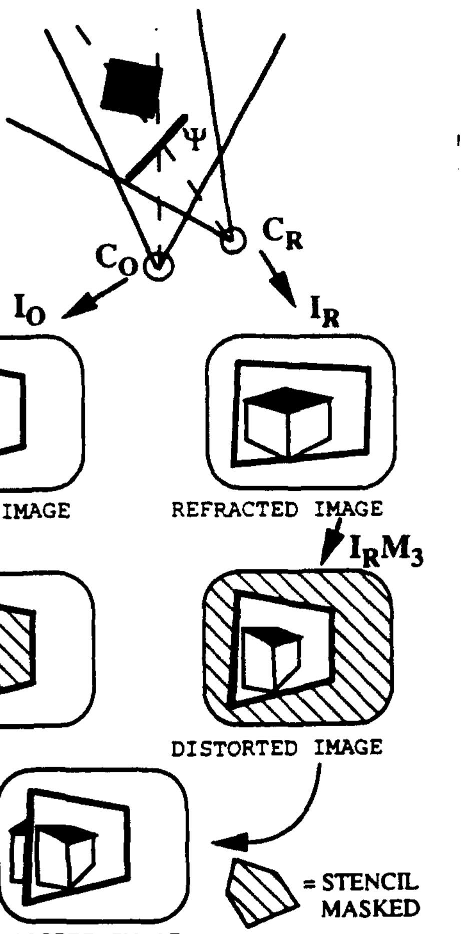


Figure 1: Refracted Image vs. Camera Image

where

$$\mathbf{M_3} = \mathbf{M_4} \cap \Psi. \tag{2}$$

and

$$\mathbf{M_4} = \mathbf{P}^{-1}\mathbf{C_r C_o}^{-1}\mathbf{P}. \tag{3}$$

In 3, $\mathbf{P}$ is the perspective transform and $\mathbf{C_o}$ and $\mathbf{C_r}$ are the original and refracted camera transforms, respectively.

This results in a 2-D projective transform of arbitrary quadrilateral to quadrilateral described in [9] and included in Appendix A. This transform, described by a 3x3 homogeneous transform, can be applied directly to the screen-viewport mapping to distort the refractive image into the normal image space. In hardware which supports user-defined transforms, this transform can be inserted directly at the end of the rendering pipeline. In systems where this is not possible, such as the Silicon Graphics^{TM} architecture, this transform can be implemented as a 4x4 homogeneous transform inserted in the world-to-unit pipeline. The resulting transform is constructed with a zero scale factor for Z so that the mapping is to the $Z = 0$ plane. Without this mapping, the tapering and skewing effects from the quadrilateral distortion affect the Z coordinates. This scaling does however preclude the use of the Z-buffer for hidden surface removal as all image points now have the same Z value. This method also does not allow for the translucency simulation described below, due to the loss of depth.

Note also that this method does not produce true refractions, merely a close approximation to the refractive image. In a true refractive image, every ray incident with the refractive plane bends according to its angle with the plane; this method, however, uses only one incident angle. In practice, two angles are used to provide more realistic results with the system. First, the incident ray is taken from the camera location to the refracting face center to determine whether the incident angle is greater than the critical angle. If this is the case, the surface is taken to be wholy reflective. If the angle is less than the critical angle, the incident angle for Snell's Law is taken at the point of intersection of the view vector (camera's negative Z axis) and the plane in which the refracting face lies. This method insures that the critical angle is reached as the plane moves tangentially to the view, yet the refracted image is seen is a smooth scrolling of the background behind the face.

5 Refractive Shadows[1]

While the above method does produce accurate reflective images and close approximations for refractive surfaces, it does not produce accurate lighting affects from these surfaces. Light reflects off a mirror and refracts through glass producing different shadows than if not present. To produce a more accurate image, these effects must also be taken into account. Therefore, any shadow generation method must not only work in conjunction with the stenciling method described above, but it must also be affected by the reflective and refractive surfaces in a scene.

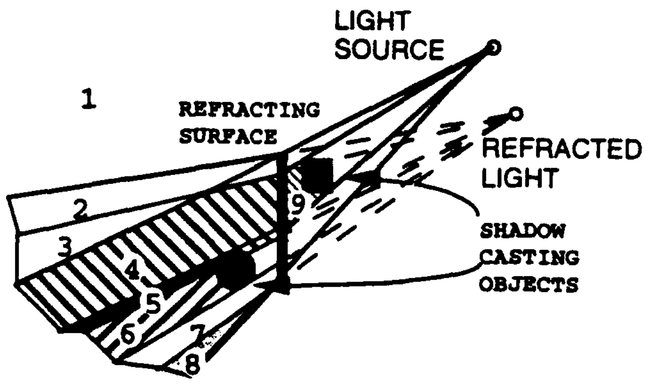
Our shadows are implemented using the traditional shadow volume technique described by [11]. This technique uses the in-out principle of silhouette faces to mask regions inside the shadow volume.

To understand how this method must be extended for refractive surfaces, examine Figure 2. This figure displays the complex shadow patterns caused by objects on both sides of a refracting surface. Note that this is *not an exact representation* but instead a hybrid model used in our system to greater demonstrate the refracting effects. The rays are refracted as in a change of medium: they do not represent true in-out refraction of a material with a thickness. With in-out refraction, the refracted rays are parallel to the incident rays and merely offset, thereby not permitting direct light to fall within the light volume. [13] Although the included images were generated with this change-of-medium model, in-out refractions are achieved merely by changing the refracting function (or by placing back-to-back refracting faces with opposing indices of refraction in the current model).

To accurately model shadows, each of the above mentioned features must be included in our shadow model. To accomplish this, we require a two-pass shadow generation approach. The first phase generates all shadows and lighting falling within the refracted light area. The second pass renders all lighting and shadows outside this area. This method creates both the shadow and caustic effects of the refractive surface.

In the first pass, a light volume [15] is generated for the refracting face. Shadow volumes are then generated

[1]Although only refractions are mentioned henceforward, the methods described are applicable for reflections with only minor variations.



1: PRIMARY LIGHT
2: PRIMARY AND REFRACTED LIGHT (NOT PRESENT WHEN EXIT REFRACTIONS)
3: CUBE'S REFRACTED SHADOW & PRIMARY LIGHT
4: CUBE'S REFRACTED SHADOW
5: CUBE'S REFRACTED SHADOW & SPHERE'S SHADOW FROM REFRACTED LIGHT
6: SPHERE'S SHADOW FROM REFRACTED LIGHT
7: REFRACTED LIGHT
8: REFRACTING SURFACE'S SHADOW

Figure 2: Light interaction with refractive surface

for shadows falling inside this volume. This itself includes two cases: namely objects inside the volume generating shadows and objects outside the volume whose shadows get refracted into the volume. In the first case, the shadow volume cannot intersect the refracting plane for to do so would place the object outside the light volume. In the second case, the shadow volume must intersect the refracting plane in order to be refracted into the light volume. Because true in-out refraction results in refracted rays parallel to incident rays, objects outside the light volume cannot cast shadows into the light volume directly from the primary light source. Both intersection cases can be checked during the shadow volume generation. A simple pre-shadow generation check using dot-products can determine if the object is on the appropriate side of the refracting plane and can save having to generate the shadow volume.

The second pass creates shadows for the entire environment. Even the refracted light volume region is included. This captures the shadow effect caused by the refractive surface itself.

6 Recursion

Both methods for rendering shadows and for rendering refraction and reflection require use of the stencil planes. While it might seem that the refraction stencil mask value would be a logical choice for the zero value in the shadow (algorithm), this is not the case. In order to have recursive refractions, we instead choose a value which is three-fourths of the maximum stencil value for our "zero" shadow value, and one-half of the maximum for our minimum shadow stencil value. This provides half of the stencil buffer for shadow calculation and half for recursive levels. These values can be

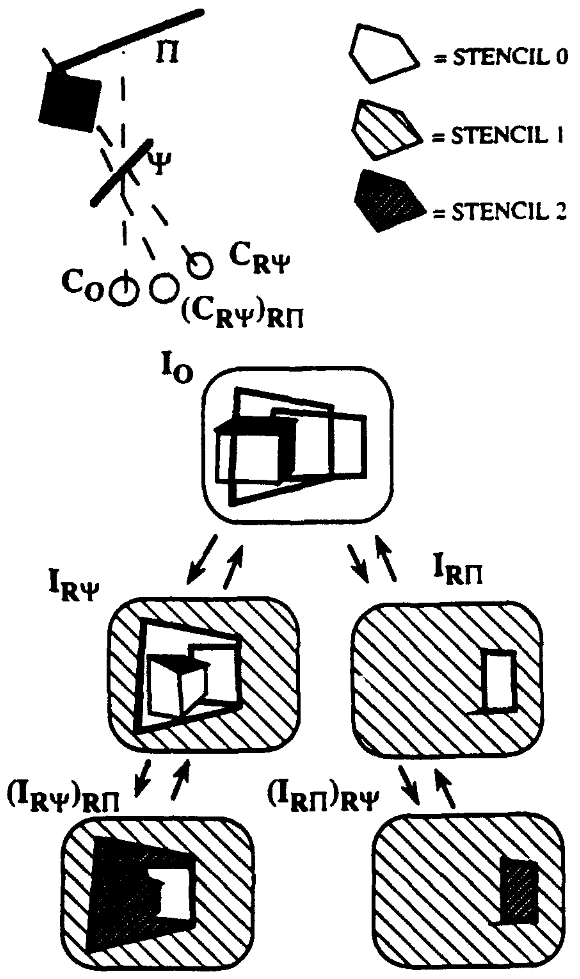


Figure 3: Recursive Stenciling

adjusted according to the recursion level needed or the shadow object complexity.

We choose zero for our render area value: this is the stencil mask value for drawing at every level of recursion. At each level of recursion, all values less than the stencil minimum are incremented by one (setting the current rendering area to one), and the new refractive surface is drawn setting the stencil value to zero. The refracted image is then drawn in the zero stencil area, and the process is repeated for all other refractive surfaces. Once the desired recursion level has been reached, all stencil values less than the shadow minimum are decremented (with zero capped), which essentially pops us back one level of recursion. The process is then repeated for the next recursive surface, with stencil values incremented by one and the surface creating a stencil mask of zero. This process is illustrated in Figure 3.

At each level of recursion, shadows must be drawn in the valid area. The reason for our choice of shadow zero is now apparent: it avoids conflict with our recursive refraction levels. All stencil values of zero at each level are changed to the shadow zero, and shadows are then rendered as described above using the in-out method. The shadow zero should be chosen so that the in-out method does not go below the shadow minimum or above the maximum stencil value in order to prevent conflict with the refraction stencils. After all shadows are drawn, all stencil values greater than shadow minimum are reset to zero, for continuation of the refraction recursion. The entire procedure is seen below:

```
drawwindow(Camera)
{
  if (SHADOWS){
    turn_lights_off();
    draw_normal_objects(Camera,ZERO); //ambient only
    for (each light){
      apply_stencil(EQUAL, ZERO, REPLACE, SHAD_ZERO);
      draw_ref_shadows();    //draw shadows in ref areas
      apply_stencil(GREATER, SHAD_MIN, REPLACE, SHAD_ZERO);
      draw_all_shadows();    //draw remaining shadows
      apply_stencil(GREATER, SHAD_MIN, REPLACE, ZERO);
    }
  }else
    draw_normal_objects(Camera,ZERO);
  if (REF){
    REF_LEVEL++;
    draw_ref_objects(Camera);
    REF_LEVEL--;
  }
}

apply_stencil(COMP_FUNC, COMP_VALUE, PASS_FUNC, PASS_VALUE)
{
  stencil(COMP_FUNC, COMP_VALUE, PASS_FUNC, PASS_VALUE);
  apply_to_screen();  //apply stencil to every pixel
}

/*SHADOW ROUTINES*/

draw_ref_shadows()
{
  for (each ref_face){
    face_light_stencil(ref_face); //light volume
    turn_light_on(light);
    make_all_shadows(MUST_INTERSECT, ref_face);
    draw_normal_objects(Camera,SHAD_ZERO);
    apply_stencil(GREATER, SHAD_MIN, REPLACE, SHAD_ZERO);
    ref_light(light,ref_face);    //move to ref position
    make_all_shadows(CAN'T_INTERSECT, ref_face);
    draw_normal_objects(Camera,SHAD_ZERO);
    apply_stencil(GREATER, SHAD_MIN, REPLACE, SHAD_ZERO);
  }
}

draw_all_shadows()
{
  make_all_shadows(ALWAYS, ref_face);
  draw_normal_objects(Camera,SHAD_ZERO);
  apply_stencil(GREATER, SHAD_MIN, REPLACE, SHAD_ZERO);
}

make_all_shadows(MODE,ref_face)
{
  for (each face)
    draw_shadow_volume(face,MODE,ref_face);
}

draw_shadow_volume()
{
  make_shadow_volume(ref_face,sv,intersect_face);
  switch (MODE){
    case CAN'T_INTERSECT:
      if (!(intersect_face==ref_face)){
        draw_shadow_stencil(sv);
        make_shadow_volume(intersect_face,MODE,ref_face);
      }
    case MUST_INTERSECT:
      if (intersect_face==ref_face){
        make_shadow_volume(intersect_face,ALWAYS,0);
      }
    case ALWAYS:
      draw_shadow_stencil(sv);
      if (intersect_face){
        make_shadow_volume(intersect_face,ALWAYS,0);
      }
}

/*REF ROUTINES (Refraction and Reflection)*/

draw_ref_objects(Camera)
{
  if (REF_LEVEL==1)
    clear_stencil(ZERO);
  apply_stencil(EQUAL, ZERO, REPLACE, REF_LEVEL);
  for (each ref_face){
    stencil(EQUAL, REF_LEVEL, REPLACE, ZERO);
    draw_face(ref_face);  //REF_LEVEL->0 where face
    RefCamera=ref_camera(ref_face,Camera);
    draw_window(REF_CAMERA);
    apply_stencil(EQUAL, ZERO, REPLACE, REF_LEVEL);
  }
  apply_stencil(EQUAL, REF_LEVEL, REPLACE, ZERO);
}

draw_normal_objects(Camera,STEN_VALUE)
{
  setup_view(Camera);     //view from camera
  stencil(EQUAL, STEN_VALUE, KEEP, STEN_VALUE);
  for (each non_ref object)
    draw_object();
  }
}
```

7 Soft Shadows and Caustics

While the above algorithm does handle the constituent effects of primary and indirect illumination, diffuse light is incrementally added resulting in producing only the umbra of the shadows. In order to produce the penumbra, an accumulation of the lighting effects from the primary as well as refracted lights must occur. Fortunately, we can use an accumulation buffer to do just this.

There are two methods for performing this accumulation of lighting effects. The first method is to treat each shadow calculation as independent and sum each resulting image. Areas which receive light from both the source and refracted light volume produce caustic effects. This method has limitations when using a single accumulation buffer due to the non-independent effects of different lights.

The second method uses an extension of shadow volumes by [3] for soft shadows. By processing all shadow volumes without producing intermediate images, the stencil value of each pixel represents a "darkness level" due to encasement in several shadow volumes. The actual lighting of the scene is performed after all shadow volumes have been generated. Assuming the darkness level is only greater than the zero stencil value, the scene is redrawn with diffuse lighting for areas whose stencil value is less than the shadow zero value. All stencil values are then decremented, and the image is redrawn. This process is repeated for each darkness level, accumulating each intermediate image. This produces a final image with intensities based on the number of enclosed shadow volumes. This method does not suffer from the problems inherent in the first method; however, extensive stencil value "juggling" of the image is necessary.

8 Object Attributes and Translucency

Refraction and reflection need not be limited to purely transparent or purely specular surfaces, respectively. We can create a multitude of materials by rendering the refractive surface again after the refractive image is drawn. This second rendering is alpha-blended with the refracted scene. On the Iris Indigo^{TM} system, this actually requires two additional renderings of the refractive face since lit faces cannot have a source alpha value. The first rendering is done without color and sets the destination alpha value to the appropriate value. The second rendering is with lighting and a blending function depending on the destination pixel chosen. This permits hardware shading effects and other hardware rendering features such as textures to be blended with the refracted scene.

In addition, translucency can be simulated using the hardware fog feature. Translucent objects act as a filter with closer objects more clearly visible than farther objects due to the random refractions which take place. This effect can be approximated using hardware fog features with the minimum fog set at the refractive plane distance and the maximum at the desired distance depending on the material property. Although fog is linear with respect to the view, the approximation is fairly accurate due to the limited angular displacement of the refracting plane due to the critical angle. The effect of translucency versus simple alpha blending is demonstrated in Figure 4.

9 Limitations and Extensions

While the system described can produce fast, complex images, it does suffer from several shortcomings. Because it relies on multiple viewpoints, refractive and reflective surfaces must be planar. Shadows suffer from aliasing effects due to the use of image space precision in the calculation. In addition, hardware shading (typically the Phong model [4]) is used for the illumination model, which is widely know for its inadequacies [19]. In the same regard, the system is hardware dependent on the number of stencil and accumulation bits, as well as the viewport-screen transforms supported.

The rendering phase is also very time-dependent on the complexity of the environment as well as the recursion level for refractions (reflections). Shadows require

$$O(num\_edges * recursive\_depth) \qquad (4)$$

for each scene rendering, of which there are

$$num\_ref\_faces*(num\_ref\_faces-1)^{(recursive\_depth-1)}. \qquad (5)$$

For complex refractive surfaces, this expense can quickly become prohibitive even when compared to raytracing.

For scenes with a limited number of planar refractive and reflective surfaces, or with a low recursive depth, this system is very effective even with minimal hardware support. The system currently runs effectively on an Iris Indigo XS24^{TM} with 8 stencil bits and a 48 bit accumulation buffer.

Additional hardware support would provide greater facilities for creating more complex images. Multiple accumulation buffers would provide greater shadow-blending capabilities. Additional pipeline control such as viewport transforms or additional fog features would enable distorted refractions in conjunction with translucency.

The method can also be extended using the accumulation buffer to handle partially reflective and partially refractive surfaces, instead of merely switching at the critical angle. Anisotropic reflections [19] could be simulated based on the reflecting plane's orientation. Speedup can be achieved by precomputation of shadow volumes during static geometric periods. Visibility checks between refractive surfaces can also be used to reduce the number of scene renderings.

10 Conclusion

We have described a series of techniques for adding realism to interactive environments and producing fast animations. These techniques have the common thread of using the hardware pipeline itself to produce illumination effects commonly found only in non-interactive renderers such as ray-tracers.

This paper is presented not only as a introduction of new methods, but to serve as a platform from which to incorporate other hardware techniques to build a complete interactive renderer, as described in [18].

More control for direct manipulation of the rendering pipeline is needed as well as more complex hardware lighting models. With advanced hardware features finding exotic uses in producing effects such as texture-mapped shadows, this stresses the need for greater flexibility in user interaction.

While there will always be a need for complex, very accurate rendering packages, many situations require fast, approximate, solutions. The techniques outlined in this paper provide such solutions for fast animation and interactive modeling.

A 2-D Quadrilateral Projective Map

The mapping of a quadrilateral to quadrilateral can be accomplished by composing the mapping of a quadrilateral-to-square with a square-to-quadrilateral. The two mappings are adjoints of each other symbolically, and therefore only the square-to-quadrilateral mapping[2] will be given:

$$\mathbf{M}_{sq} = \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}, \qquad (6)$$

where

$$\begin{aligned} g &= \begin{vmatrix} \sum x & \delta x_2 \\ \sum y & \delta y_2 \end{vmatrix} \Big/ \begin{vmatrix} \delta x_1 & \delta x_2 \\ \delta y_1 & \delta y_2 \end{vmatrix} \\ h &= \begin{vmatrix} \delta x_1 & \sum x \\ \delta y_1 & \sum y \end{vmatrix} \Big/ \begin{vmatrix} \delta x_1 & \delta x_2 \\ \delta y_1 & \delta y_2 \end{vmatrix} \end{aligned} \qquad (7)$$

$$\begin{aligned} a &= x_1 - x_0 + gx_1 & d &= y_1 - y_0 + gy_1 \\ b &= x_3 - x_0 + hx_3 & e &= y_3 - y_0 + hy_3 \\ c &= x_0 & f &= y_0 \end{aligned} \qquad (8)$$

[2]This mapping is also equal to the perspective transform of the camera rotation with respect to the normal of the refracting plane, which is available directly from the matrix stack

B 3-D Quadrilateral Projective Map

Given the 2-D quadrilateral projective transform

$$\mathbf{M}_{qq_3} = \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}, \qquad (9)$$

we create the 3-D transform

$$\mathbf{M}_{qq_4} = \begin{bmatrix} a & d & 0 & g \\ b & e & 0 & h \\ 0 & 0 & 0 & 0 \\ c & f & 0 & i \end{bmatrix}, \qquad (10)$$

which clears depth values and disables Z-buffering and fog.

References

[1] John M. Airey, John H. Rohlf, and Fredrick P. Brooks. Towars image realism with interactive update rates in complex virtual building environments. *ACM SIGGRAPH Special Issue on 1990 Symposium on Interactive 3D Graphics*, 24(2):41–50, 1990.

[2] J. Arvo. Backward ray tracing, developments in ray tracing. *SIGGRAPH Course Notes*, 12, 1986.

[3] L. Brotman and N. Badler. Generating soft shadows with a depth buffer algorithm. *IEEE Computer Graphics and Applications*, 4(10):71–81, 1984.

[4] Phong Bui-Thong. Illumination for computer-generated pictures. *Communications of the ACM*, 18(6), 1975.

[5] S. Chattopadhyay and A. Fujimoto. Bi-directional ray tracing. *Porc. of CG International '87*, 1987.

[6] S.E. Chen. Incremental radiosity: An extension of progressive radiosity to an interactive image synthesis system. *ACM Computer Graphics (Proc. SIGGRAPH '90)*, 24(4), 1990.

[7] J. Dorsey. *Computer Graphics Techniques for Opera Lighting Design and Simulation*. PhD thesis, Cornell University, 1993.

[8] A.S. Glassner, editor. *An Introduction to Ray Tracing*. Academic Press, London, 1989.

[9] P.S. Heckbert. Fundamentals of texture mapping and image warping. Master's thesis, University of California, Berkley, 1989.

[10] P.S. Heckbert and P. Hanrahan. Beam tracing polygonal objects. *ACM Computer Graphics (Proc. SIGGRAPH '84)*, 18(3), 1984.

[11] T. Heidmann. Real shadows - real time. *IRIS Universe*, November-December 1991.

[12] J. Kajiya. The rendering equation. *ACM Computer Graphics (Proc. SIGGRAPH '86)*, 20(4):143–150, 1986.

[13] D.S. Kay and D. Greenberg. Transparency for computer synthesized images. *ACM Computer Graphics (Proc. SIGGRAPH '79)*, pages 158–164, 1979.

[14] E. C. Kingsley, N.A. Schofield, and K. Case. Sammie. *ACM Computer Graphics (Proc. SIGGRAPH '81)*, 15(3), 1981.

[15] T. Nishita. A shading model for atmospheric scattering considering luminous intensity distribution of light sources. *ACM Computer Graphics (Proc. SIGGRAPH '87)*, 19(3):303–310, 1987.

[16] C.H. Sèquin and E.K. Smyrl. Parameterized ray tracing. *ACM Computer Graphics (Proc. SIGGRAPH '89)*, 23(4):307–314, 1989.

[17] S.J. Teller and C.H. Sèquin. Visibility preprocessing for interactive walkthroughs. *ACM Computer Graphics (Proc. SIGGRAPH '91)*, 25(4):61–69, 1991.

[18] P. J. W. ten Hagen, A. A. M. Kujik, and C. G. Trienekens. Display architecture for VLSI-based graphics workstations. *Advances in Computer Graphics Hardware I, Record of First Eurographics Workshop on Graphics Hardware*, pages 3–16, 1986.

[19] G.J. Ward. A ray tracing solution for diffuse interreflection. *ACM Computer Graphics (Proc. SIGGRAPH '88)*, 22(4):85–92, 1988.

[20] A. Watt and M. Watt. *Advanced Animation and Rendering Techniques*. Adderson/Wesley, 1992.