REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 22 May 2002	3. REPORT TYPE AND DATES COVERED Final 29 Sep 2000 – 31 Mar 2002

4. TITLE AND SUBTITLE

Basic Skills Trainer Simulation Improvements

5. FUNDING NUMBERS

6. AUTHORS

Stephen J. Dow

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

The University of Alabama in Huntsville
Huntsville, AL 35899

8. PERFORMING ORGANIZATION REPORT NUMBER

5-21171

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Army Aviation & Missile Command
Redstone Arsenal, AL 35898

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The set of SED Basic Skills Trainers use common software for moving 3D target models over a background terrain. This report documents several improvements made to that software to provide greater realism, specifically antialiasing to smooth the borders between target and terrain, target hazing to better color-correct the targets based on their range, and improvements to the target orientation interpolation to provide smooth turns.

14. SUBJECT TERMS
Target rendering, antialiasing

15. NUMBER OF PAGES
5

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	

Basic Skills Trainer Simulation Improvements

Stephen J. Dow
Department of Mathematical Sciences
The University of Alabama in Huntsville

May 22, 2002

Final Report

F/DOD/ARMY/AMCOM/Basic Skills Trainer Simulation Improvements
DAAH01-97-D-R005 D.O. 29

Period of Performance: 9/29/00 to 3/31/02

1. Introduction

The set of SED Basic Skills Trainers use common software for moving 3D target models over a background terrain. This report documents several specific improvements made to that software to provide greater realism. Section 2 document procedures added to the target rendering system to smooth the border between target and terrain and adjust target color as a function of range. Section 3 documents procedures added to improve the way targets move by smoothing their turns.

2. Target Antialiasing and Hazing

To explain the improvements in the target rendering it will be helpful to begin with a brief overview of the system used to render targets and combine them into a terrain scene. Each target is associated with a target model, which is the data structure containing vertices and polygonal faces describing the geometry, together with texture files, and color information and/or texture coordinates which determine what is drawn within each face. The target model is stored on disk as a VRML file, which is read into memory, along with the associated texture files, at initialization. In addition each target is associated with a path, which determines the position and orientation of the target at any given time during the simulation. Figure 1 illustrates the portion of the simulation display path dealing with the display of targets; the following three steps are labeled in the diagram:

1) A target model is rendered into the common rendering buffer;

2) The contents of the rendering buffer are transferred to a buffer specific to the target;

3) The rendered target buffers are combined with terrain imagery to produce a portion of the displayed scene.

The rendering (step 1) is accomplished using Direct3D. A Direct3D rendering buffer in video memory is set up at initialization as the intermediate location into which all targets are rendered. During an iteration of the main simulation loop, the current position and orientation of each target is updated, and expressed in the form of a 4 x 4 matrix defining the transformation from local target coordinates to terrain image coordinates. This transformation is

applied to each of the target vertices to produce a bounding box for the rendered target, consisting of a bounding rectangle, named `rcurr`, in terrain image coordinates together with bounds for the transformed z-coordinates, called `zmin` and `zmax`. If `rcurr` intersects the terrain rectangle currently being viewed (i.e. the target is in the gunner's field-of-view), and the target has undergone some change since it was last rendered, then the target is rendered anew. The transformation matrix is adjusted so that the upper left corner of `rcurr` will land at the upper left corner of the rendering buffer. The rendering buffer is filled with a special background color not expected to occur within the target. The target vertices are then transformed and passed , along with color and texture information, to the Direct3D rendering procedure.

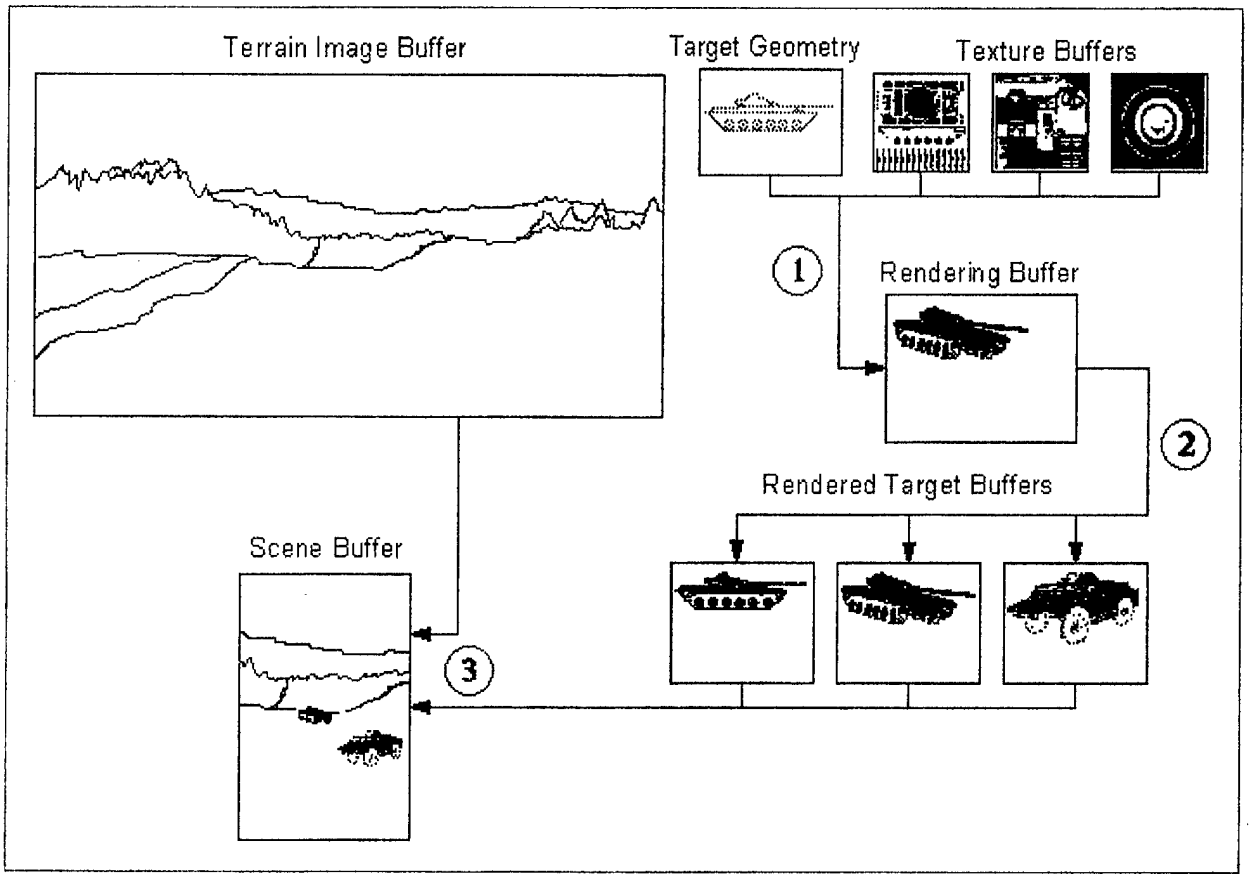

Figure 1: Target Rendering Process

Step 2 of the process simply copies the newly rendered target from the rendering buffer to the target buffer, making the rendering buffer available for use in rendering the next target. After this step the target buffer contains the special background color at each nontarget pixel, and some other color at each target pixel; also the upper left corner of `rcurr` indicates where in the terrain image the rendered target belongs, and a range value has been stored for the target. After step 2 is completed for all targets, Step 3 begins by filling the scene buffer with terrain pixels, and extracting a corresponding buffer of range values from the full range image. Then, stepping through the list of targets in the field-of-view, and stepping through each pixel in the corresponding `rcurr`, the pixel is copied from

the target buffer to the scene buffer, and the range buffer updated at that pixel, if the pixel is a target pixel (not the background color) and the target range is smaller than the current range at that pixel (target is not occluded).

The procedure just described was the one used in the initial versions of the BST software. It is based on specifying each pixel as completely target or terrain. The effect, especially for small targets consisting of only a few pixels, is a noticeable jagged edge along the target/terrain boundary. As the target moves, this boundary often proceeds in a "walking" fashion; for example if the target is approximately three pixels high and is moving mostly horizontally, one may see the bottom row of pixels shift one pixel sideways first, then the middle row shift to join the bottom row, then the top row shift to join the other two, then the bottom row proceed again, etc.

Antialiasing is a standard procedure for reducing this jagged boundary effect. We have implemented it as follows. After computing `rcurr` as described above, we compute an integer scale factor `s` based on the ratio of the size of the rendering buffer to the size of `rcurr`. The transformation matrix is scaled by `s`, causing the rendered target to be magnified. In this magnified image of the target, each block of `s x s` pixels corresponds to a single pixel in the unmagnified target. The magnified target image is read from the rendering buffer into a temporary buffer, and is a 24-bit RGB image. As before, nontarget pixels in that image are identifiable as those with the special background color. In transferring this image to the target buffer, we scale it back down and in the process convert it to 32-bit RGBA (red, green, blue, alpha) format. Each pixel in the new image is computed from its corresponding `s x s` block in the magnified image, as follows. The R, G, and B values are computed as the average of the corresponding values of the target pixels in the `s x s` block (if there are no target pixels, we assign each component a value of 0). The alpha value is computed as A = 255 * `count/(s * s)`, where `count` is the number of target pixels in the block. Thus A = 0 corresponds to a completely background pixel, A = 255 to a completely target pixel, and values between to partially target pixels.

Step 3 in the diagram is modified to incorporate antialiasing by blending the target and terrain pixel values according to the alpha values. Thus if a target pixel passes the occlusion test (target range less than range at that pixel in the range buffer), then the scene pixel is computed as

```
R = (A * R1 + (255 - A) * R2)/255
G = (A * G1 + (255 - A) * G2)/255
B = (A * B1 + (255 - A) * B2)/255
```

where (R1, G1, B1, A) is the value found in the target buffer, and (R2, G2, B2) the value previously in the scene buffer at that pixel.

The other improvement made to the display of targets is called target hazing. Following Step 2, target hazing is applied to the image stored in the target buffer. It uses the same algorithm that is applied to the whole scene to simulate fog, but is applied in addition to any other such weather effect. The algorithm consists of replacing a given pixel with a weighted average of its color value and a fog/haze color value

```
c_new = (1 - w) * c_old + w * c_fog
```

The weight factor `w` is range dependent, but since the whole target is regarding as being at a single range, `w` is constant for each target. The range dependence is formulated in terms of range values `r0` and `r1`, and a maximum weight factor `wmax`. For `r < r0, w = 0` (no fogging). For `r > r1, w = wmax` (maximum fogging). For

values of r between r0 and r1, w is computed by linear interpolation. Having computed w, a lookup table is computed and used to apply the weighted averaging in the equation above to the target pixels in the target buffer. The alpha values in the target buffer are left unchanged in this process.

3. Smooth Target Turns

This section documents the method by which target orientation is calculated as a target moves along its path; the method having been improved from the first versions of the software. Each target is associated with a target path consisting of a sequence of locations (vertices) in ground coordinates for a target to follow. Targets and their paths are designated as ground or aerial. For a ground path, all locations where the path crosses an edge of the ground triangulation are included as vertices. The portion of a path between consecutive vertices is called a path segment. A time is given in the path file for each segment, specifying how long the target will spend on that segment. Stationary segments are allowed; i.e. consecutive stored vertices at the same ground coordinates. With each path segment we associate a rotation from target to ground coordinates. For stationary segments, this rotation is specified in the path file; for other segments it is computed so as to orient the target in the direction of the path segment, with the vertical axis of the target oriented either normal to the ground triangle on which that segment lies, for ground targets, or simply in the "up" direction of the ground coordinate system, for aerial targets. These rotations are initially computed as 3 x 3 matrices, and are then converted to quaternions, which allows us to interpolate between them. Thus at initialization, we compute and store a quaternion for each path vertex, incorporating the direction from that vertex to the next.

To interpolate orientations during the simulation, we use a function prototyped as

```
void MaInterpolateQuaternion(Quaternion *q1, Quaternion *q2, double t,
                             Quaternion *q)
```

Inputs to this function are quaternions q1 and q2 and a parameter t; the output quaternion q will equal q1 when t = 0, it will equal q2 when t = 1, and it will take an intermediate value when t lies between 0 and 1. Within an iteration of the simulation, each target's orientation is updated as follows. The current time (in milliseconds from simulation startup) is stored as tc. From this time, and the segment times associated with each path segment, a current vertex is determined, such that the target is within the path segment starting at current vertex and proceeding toward the next vertex. The time it reached the current vertex is stored as ts, which we refer to as the vertex start time. These times satisfy the relation

$$ts \leq tc < ts + seg\_time,$$

where seg_time is the time assigned to the current segment.

During an interval of time extending before and after a target reaches a given vertex, the target's orientation is interpolated between the precomputed orientations associated with the preceding and following segments. We have arbitrarily defined that interval to extend from one second before to one second after the vertex start time, unless either segment is less than two seconds long, in which case the interval to that side of the vertex is reduced to

half the segment time. Thus we compute the time interval `ti` over which interpolation will occur for the current segment as

```
ti = 1000;
if (ti > seg_time / 2)
      ti = seg_time / 2;
```

Now, if `tc < ts + ti`, then the target is in the initial portion of the current segment, where its orientation is interpolated from the orientations `q1` and `q2` of the preceding and current segments. Parameter t is computed as

```
t = 0.5 + (tc - ts)/(2*ti);
```

Thus when `tc = ts`, which is when the target has just reached the current vertex, t = 0.5, resulting in an orientation exactly halfway between those associated with the segments to either side of that vertex. In the case `tc > ts + vert->seg_time - ti`, the target is in the tail end of the current segment, where its orientation is interpolated from the orientations `q1` and `q2` of the current and following segments. Parameter t is computed as

```
t = (tc - (ts + seg_time - ti))/(2*ti);
```

When `tc = ts + seg_time`, which is when the target will reach the next vertex, the formula again gives t = 0.5. For values of `tc` between `ts + ti` and `ts + seg_time - ti`, the quaternion associated with the current segment is used without interpolation, the orientation being left unchanged during that middle portion of the segment.

An exception is made to the computations above in the case of a *stationary* segment, where the ground position is the same at two successive vertices. The orientation is allowed to change during such a segment (as for a tank rotating in place); in this case the stored orientations are treated as those desired at the two vertices (i.e. endpoints of the time interval during which the rotation takes place). In this case, `q1` and `q2` are assigned to be the quaternions stored with those two vertices, and t is computed as

```
t = (tc - ts)/seg_time;
```

The interpolation thus occurs throughout the path segment, and `ti` is not used in the computations.