ARMY RESEARCH LABORATORY

Three-Dimensional Translations, Rotations, and Intersections Using C++

by Robert J. Yager

ARL-TN-557 **August 2013**

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5066

ARL-TN-557 **August 2013**

Three-Dimensional Translations, Rotations, and Intersections Using C++

Robert J. Yager
Weapons and Materials Research Directorate, ARL

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
August 2013	Final	April 2012–May 2012

4. TITLE AND SUBTITLE
Three-Dimensional Translations, Rotations, and Intersections Using C++

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Robert J. Yager

5d. PROJECT NUMBER
AH80

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: RDRL-WML A
Aberdeen Proving Ground, MD 21005-5066

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TN-557

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

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

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

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Three-dimensional (3-D) operations, such as rotations, translations, and intersections, are tools that are essential for many types of scientific modeling. However, the C++ programming language does not natively perform them. This report documents a set of functions, written in C++, that can be used to perform 3-D rotations, translations, and intersections.

15. SUBJECT TERMS
rotation, translation, intersection, line, plane, triangle, point, three-dimensional, C++, operation

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Robert J. Yager
Unclassified	Unclassified	Unclassified	UU	22	19b. TELEPHONE NUMBER (*Include area code*) 410-278-6689

Contents

List of Figures

Acknowledgments

The author would like to thank Mr. Luke Strohm and Mr. Benjamin Flanders of the U.S. Army Research Laboratory's Weapons and Materials Research Directorate. Mr. Strohm provided technical and editorial recommendations that improved the quality of this report. Mr. Flanders provided insight into how the C++ functions presented in this report could best be generalized.

INTENTIONALLY LEFT BLANK.

1. Introduction

Three-dimensional (3-D) operations, such as rotations, translations, and intersections, are tools that are essential for many types of scientific modeling. However, the C++ programming language does not natively perform them. This report documents a set of functions, written in C++, that can be used to perform 3-D rotations, translations, and intersections. All of the functions have been grouped together in the namespace y3DOps. A summary sheet that reproduces the entire y3DOps namespace is located at the end of this report.

2. Translation of a Point in Space

2.1 Derivation

Let the position vector $\vec{p}$ represent an arbitrary point in space, where

$$\vec{p} = p_x\hat{x} + p_y\hat{y} + p_z\hat{z}\,. \tag{1}$$

Furthermore, let $\vec{d}$ represent a displacement vector, where

$$\vec{d} = d_x\hat{x} + d_y\hat{y} + d_z\hat{z}\,. \tag{2}$$

If $\vec{p}\,'$ is used to represent $\vec{p}$ after it has been translated, then

$$\vec{p}\,' = (p_x + d_x)\hat{x} + (p_y + d_y)\hat{y} + (p_z + d_z)\hat{z}\,. \tag{3}$$

2.2 C++ Implementation

Translate() Code

```
inline void Translate(//<================================PERFORMS A TRANSLATION
    double p[3],//<-------COORDINATES TO TRANSLATE (MODIFIED BY THIS FUNCTION)
    const double d[3]){//<---------------------------------DISPLACEMENT VECTOR
  p[0]+=d[0] , p[1]+=d[1] , p[2]+=d[2];
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
```

Translate() Parameters

p — **p** is a three-element array that stores the position vector that is described by equation 1. Note that **p** is modified by the Translate() function, as described by equation 3.

d — **d** is a three-element array that stores the displacement vector that is described by equation 2. **d** determines the amount and direction by which **p** is translated.

Translate() Example

Figure 1 shows point $\vec{p}$ being translated to a new position ($\vec{p}'$) by displacement vector $\vec{d}$.

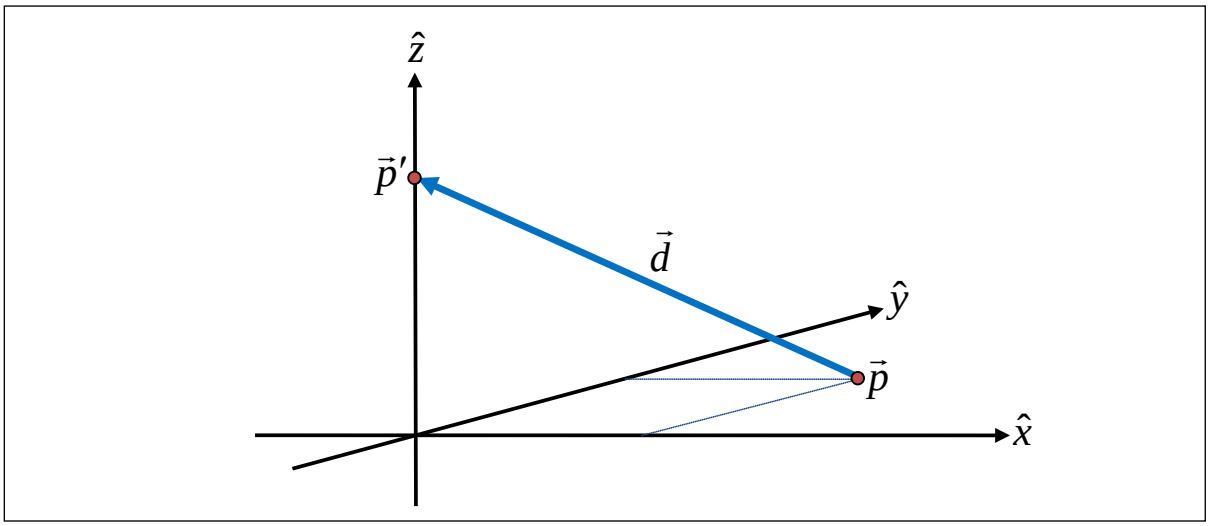


Figure 1. Translate() example.

Let $\vec{p} = \{1,1,0\}$ and $\vec{d} = \{-1,-1,2\}$. Point $\vec{p}'$ can be found by using the Translate() function, as shown in the following sample code.

```
#include <cstdio>//.....................................................printf()
#include "y_3d_ops.h"//.............................................Translate()
int main(){
  double p[3]={1,1,0};
  double d[3]={-1,-1,2};
  y3DOps::Translate(p,d);
  printf("p[0]=%f, p[1]=%f, p[2]=%f\n",p[0],p[1],p[2]);
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
```

OUTPUT:

```
p[0]=0.000000, p[1]=0.000000, p[2]=2.000000
```

---

3. Rotation of a Point About an Arbitrarily Positioned Axis

3.1 Derivation

Suppose that the unit vector $\hat{v}$ is used to define an arbitrary axis about which a point in space will be rotated.

$$\hat{v} = v_x\hat{x} + v_y\hat{y} + v_z\hat{z}\,. \tag{4}$$

Rodrigues' rotation formula can be used to construct a rotation matrix,[1] R, that can be used to perform a rotation about $\hat{v}$ by an angle θ. Note that the direction of the rotation will be determined by the right-hand-thumb rule (when the right thumb is pointed in the direction of $\hat{v}$, the curled fingers of the right hand will point in the direction of the rotation).

$$R = \begin{bmatrix} v_x^2(1-c_\theta)+c_\theta & v_xv_y(1-c_\theta)-v_zs_\theta & v_xv_z(1-c_\theta)+v_ys_\theta \\ v_xv_y(1-c_\theta)+v_zs_\theta & v_y^2(1-c_\theta)+c_\theta & v_yv_z(1-c_\theta)-v_xs_\theta \\ v_xv_z(1-c_\theta)-v_ys_\theta & v_yv_z(1-c_\theta)+v_xs_\theta & v_z^2(1-c_\theta)+c_\theta \end{bmatrix} \tag{5}$$

where

$$c_\theta \equiv \cos(\theta) \tag{6}$$

and

$$s_\theta \equiv \sin(\theta). \tag{7}$$

Let the position vector $\vec{p}$ locate an arbitrary point in space, where

$$\vec{p} = p_x\hat{x} + p_y\hat{y} + p_z\hat{z}\,. \tag{8}$$

Let the position vector $\vec{o}$ locate the origin of the rotation axis defined by $\hat{v}$.

$$\vec{o} = o_x\hat{x} + o_y\hat{y} + o_z\hat{z}\,. \tag{9}$$

The translation-rotation-translation sequence described by equation 10 can be used to find $\vec{p}\,'$, where $\vec{p}\,'$ is used to represent $\vec{p}$ after it has been rotated about $\hat{v}$.

$$\vec{p}\,' = R(\vec{p} - \vec{o}) + \vec{o}\,. \tag{10}$$

3.2 C++ Implementation

Two functions are used to perform 3-D rotations. The first function, RMatrix(), calculates the rotation matrix that is presented in equation 5. The second function, Rotate(), performs the rotation that is presented in equation 10. Breaking the calculation into two functions allows functions that rotate objects containing more than one point to be written in a manner that doesn't sacrifice performance.

A special note about rotations – Although the rotation functions presented in this report were designed to be used to rotate position vectors, they can also be used with other types of vectors. For example, suppose that a multi-point, rigid object has a velocity vector associated with it.

---

[1] Mason, M. T. *Mechanics of Robotic Manipulation*; Massachusetts Institute of Technology: United States, 2001, (page 46, equation 3.26).

That object can be rotated by first creating a rotation matrix using the RMatrix() function. That rotation matrix can then be supplied to the Rotate() function to rotate all of the object's points. Using the same rotation matrix, the Rotate() function can also be used to rotate the object's velocity vector. However, the origin (i.e., the second argument of the Rotate() function) will need to be set to {0,0,0} for the call that rotates the velocity vector. Other types of vector quantities, such as torque, angular velocity, and rotation axes can similarly be rotated. However, the origin will typically need to be zeroed for vectors that are not displacement vectors.

RMatrix() Code

```
inline void RMatrix(//<=======================CALCULATES A 3D ROTATION MATRIX
    double R[9],//<--------------ROTATION MATRIX (CALCULATED BY THIS FUNCTION)
    const double v[3],//<-------THE AXIS OF ROTATION (V MUST BE A UNIT VECTOR)
    double rads){//<--THE ANGLE OF ROTATION (CCW ABOUT V, USE RIGHT-HAND RULE)
  double c=cos(rads),d=1-c,s=sin(rads),v0=v[0],v1=v[1],v2=v[2];
  R[0]=v0*v0*d+   c , R[1]=v0*v1*d-v2*s , R[2]=v0*v2*d+v1*s ;
  R[3]=v1*v0*d+v2*s , R[4]=v1*v1*d+   c , R[5]=v1*v2*d-v0*s ;
  R[6]=v2*v0*d-v1*s , R[7]=v2*v1*d+v0*s , R[8]=v2*v2*d+   c ;
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
```

RMatrix() Parameters

R **R** is a nine-element array that stores the rotation matrix that is described by equation 5. Note that **R** is modified by the RMatrix() function. **R** is intended to be used as the third argument of the Rotate() function.

v **v** is a three-element array that stores the rotation axis that is described by equation 4. **Note that v must be a unit vector.**

rads **rads** is used to represent the angle (in radians) of the rotation. As described in section 3.1, the direction of the rotation is determined by the right-hand-thumb rule.

Rotate() Code

```
inline void Rotate(//<====================================PERFORMS A ROTATION
    double p[3],//<----------COORDINATES TO ROTATE (MODIFIED BY THIS FUNCTION)
    const double o[3],//<------------------THE ORIGIN OF THE AXIS OF ROTATION
    const double R[9]){//<-----------------A ROTATION MATRIX (FROM RMatrix())
  double t0=p[0]-o[0],t1=p[1]-o[1],t2=p[2]-o[2];//.....translate to the origin
  p[0]=R[0]*t0+R[1]*t1+R[2]*t2+o[0];//.............................rotate, then
  p[1]=R[3]*t0+R[4]*t1+R[5]*t2+o[1];//                                  reverse
  p[2]=R[6]*t0+R[7]*t1+R[8]*t2+o[2];//                              translation
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
```

Rotate() Parameters

p **p** is a three-element array that stores the position vector that is described by equation 8. Note that **p** is modified by the Rotate() function, as described by equation 10.

o **o** is a three-element array that stores the position vector that is described by equation 9. **o** is the origin of **v**, the axis about which **p** is rotated.

R **R** is a rotation matrix that has been precalculated using the RMatrix() function.

Rotate() Example

Figure 2 shows point $\vec{p}$ being rotated about the axis $\hat{v}$ to a new position ($\vec{p}'$).

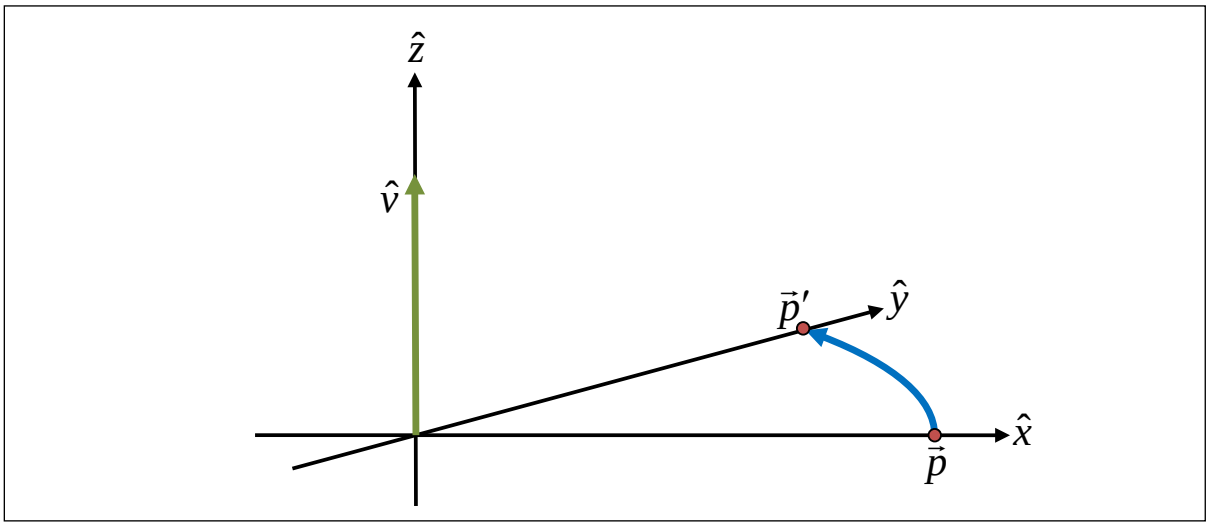


Figure 2. Rotate() example.

Let $\vec{p} = \{2,0,0\}$ and $\hat{v} = \{0,0,1\}$. Furthermore, let the origin of $\hat{v}$ be at $\vec{o} = \{0,0,0\}$, and the angle of rotation be $\pi/2$. Point $\vec{p}'$ can be found by first using the RMatrix() function to calculate a rotation matrix, then using the Rotate() function to perform the rotation.

```
#include <cstdio>//.....................................................printf()
#include "y_3d_ops.h"//.....................................RMatrix(),Rotate()
int main(){
  double p[3]={2,0,0};
  double v[3]={0,0,1};//.........................note v must be a unit vector
  double o[3]={0,0,0};
  double R[9];/*<-*/y3DOps::RMatrix(R,v,3.14159265358979/2);//...rotation matrix
  y3DOps::Rotate(p,o,R);
  printf("p[0]=%f, p[1]=%f, p[2]=%f\n",p[0],p[1],p[2]);
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
```

OUTPUT:

```
p[0]=0.000000, p[1]=2.000000, p[2]=0.000000
```

4. Intersection Between a Line and a Plane

4.1 Derivation

Suppose that a line S passes through the points $\vec{S}_0$ and $\vec{S}_1$, where

$$\vec{S}_0 = S_{0,x}\hat{x} + S_{0,y}\hat{y} + S_{0,z}\hat{z} \text{ and } \vec{S}_1 = S_{1,x}\hat{x} + S_{1,y}\hat{y} + S_{1,z}\hat{z}. \tag{11}$$

Let $\vec{p}_S$ represent a point that lies on S. $\vec{S}_0$ and $\vec{S}_1$ can be used to construct a parametric equation for $\vec{p}_S$ as a function of the parameter t_0:

$$\vec{p}_S = \vec{S}_0 + \left(\vec{S}_1 - \vec{S}_0\right)t_0. \tag{12}$$

The parameter t_0 represents the scaled distance from $\vec{S}_0$ to $\vec{S}_1$ along S. Thus if $t_0 = 0$, $\vec{p}_S$ is located at $\vec{S}_0$. If $t_0 = 1$, $\vec{p}_S$ is located at $\vec{S}_1$.

Next, suppose that a plane T passes through the points $\vec{T}_0$, $\vec{T}_1$, and $\vec{T}_2$, where

$$\vec{T}_0 = T_{0,x}\hat{x} + T_{0,y}\hat{y} + T_{0,z}\hat{z}\ ,\ \vec{T}_1 = T_{1,x}\hat{x} + T_{1,y}\hat{y} + T_{1,z}\hat{z}, \text{ and } \vec{T}_2 = T_{2,x}\hat{x} + T_{2,y}\hat{y} + T_{2,z}\hat{z}. \tag{13}$$

Let $\vec{p}_T$ represent a point that lies on T. $\vec{T}_0$, $\vec{T}_1$, and $\vec{T}_2$, can be used to construct a parametric equation for $\vec{p}_T$ as a function of the parameters t_1 and t_2:

$$\vec{p}_T = \vec{T}_0 + \left(\vec{T}_1 - \vec{T}_0\right)t_1 + \left(\vec{T}_2 - \vec{T}_0\right)t_2. \tag{14}$$

Similar to t_0, t_1 and t_2 represent scaled distances from $\vec{T}_0$ to $\vec{T}_1$ and from $\vec{T}_0$ to $\vec{T}_2$, respectively.

The point of intersection between S and T occurs where $\vec{p}_S$ is equal to $\vec{p}_T$. Thus,

$$\vec{S}_0 + \left(\vec{S}_1 - \vec{S}_0\right)t_0 = \vec{T}_0 + \left(\vec{T}_1 - \vec{T}_0\right)t_1 + \left(\vec{T}_2 - \vec{T}_0\right)t_2. \tag{15}$$

Rearranging terms,

$$\vec{S}_0 - \vec{T}_0 = \left(\vec{S}_0 - \vec{S}_1\right)t_0 + \left(\vec{T}_1 - \vec{T}_0\right)t_1 + \left(\vec{T}_2 - \vec{T}_0\right)t_2. \tag{16}$$

This can be written in matrix form as

$$\begin{bmatrix} S_{0,x} - T_{0,x} \\ S_{0,y} - T_{0,y} \\ S_{0,z} - T_{0,z} \end{bmatrix} = \begin{bmatrix} S_{0,x} - S_{1,x} & T_{1,x} - T_{0,x} & T_{2,x} - T_{0,x} \\ S_{0,y} - S_{1,y} & T_{1,y} - T_{0,y} & T_{2,y} - T_{0,y} \\ S_{0,z} - S_{1,z} & T_{1,z} - T_{0,z} & T_{2,z} - T_{0,z} \end{bmatrix} \begin{bmatrix} t_0 \\ t_1 \\ t_2 \end{bmatrix}. \tag{17}$$

Solving for $\vec{t}$,

$$\begin{bmatrix} t_0 \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} S_{0,x} - S_{1,x} & T_{1,x} - T_{0,x} & T_{2,x} - T_{0,x} \\ S_{0,y} - S_{1,y} & T_{1,y} - T_{0,y} & T_{2,y} - T_{0,y} \\ S_{0,z} - S_{1,z} & T_{1,z} - T_{0,z} & T_{2,z} - T_{0,z} \end{bmatrix}^{-1} \begin{bmatrix} S_{0,x} - T_{0,x} \\ S_{0,y} - T_{0,y} \\ S_{0,z} - T_{0,z} \end{bmatrix}. \quad (18)$$

Recall that the vector $\vec{t}$ contains the parameters from equations 12 and 14. Thus, once $\vec{t}$ is known, t_0 can be substituted into equation 12 to find the point of intersection between S and T. Note that if the three-by-three matrix defined in equation 18 is noninvertible, then S is parallel to T.

Equation 14 can be used to find a set of conditions that test whether S intersects T inside or outside the triangle defined by the three points $\vec{T}_0$, $\vec{T}_1$, and $\vec{T}_2$. Note the colored bands shown in figure 3. The blue band represents the region where $0 \le t_1 \le 1$, and the yellow band represents the region where $0 \le t_2 \le 1$.

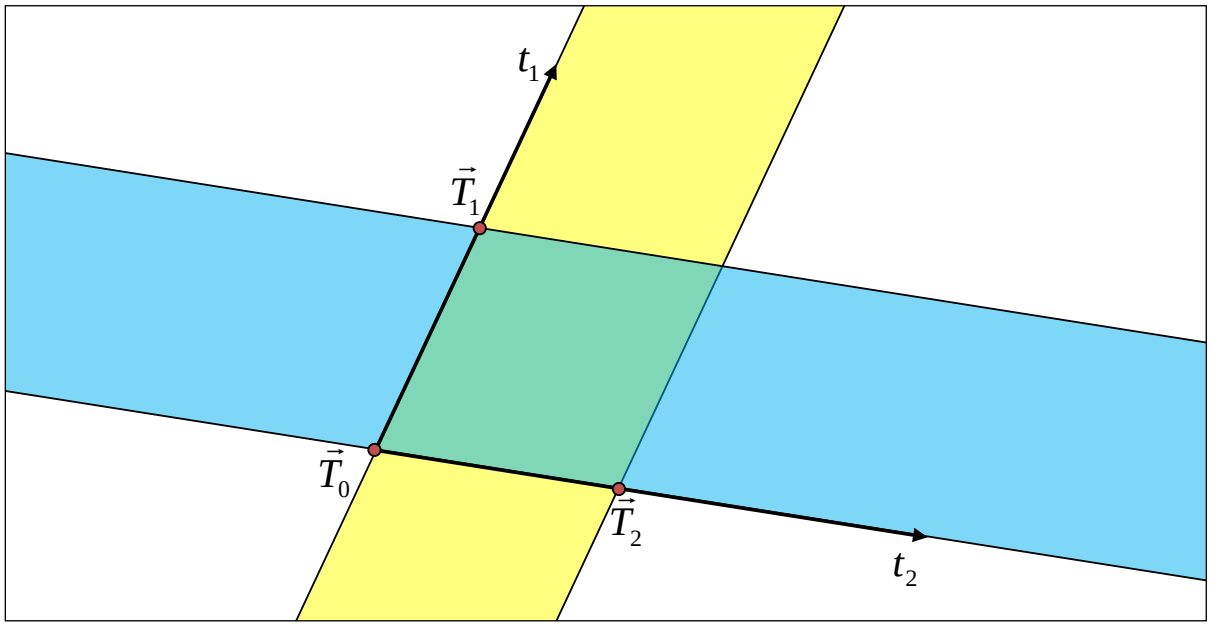


Figure 3. Parallelogram defined by the three points that make up *T*.

The region where the two bands overlap is where both sets of conditions are met. Thus, the conditions presented in equation 19 can be used to determine whether or not line S intersects the plane inside the parallelogram shown in green in figure 3.

$$0 < t_1 < 1 \text{ and } 0 < t_2 < 1. \quad (19)$$

The additional test shown in equation 20 can be used to determine whether or not line S intersects the plane inside the triangle defined by the three points $\vec{T}_0$, $\vec{T}_1$, and $\vec{T}_2$.

$$(t_1 + t_2) < 1. \tag{20}$$

Finally, if the condition presented in equation 21 is met, then S intersects T between S_0 and S_1.

$$0 \le t_0 \le 1. \tag{21}$$

4.2 C++ Implementation

Two functions are used to find line-plane intersections. The first function, IParameters(), calculates a three-element array that is the solution to equation 18. The second function, Intersect(), calculates the point of intersection between a line and a plane.

Because there is a chance that the three-by-three matrix shown in equation 18 will be singular, a Boolean that indicates whether or not a solution is valid is returned by the IParameters() function. The intersection tests described in equation 19 are performed by the Intersect() function.

Note that the Intersect() function tests for intersection between a parallelogram and a line. If a test for intersection between a *triangle* and a line is desired, then the additional condition presented in equation 20 must be tested for. If a test for intersection between a triangle and a line *segment* is desired, then the additional condition presented in equation 21 must be tested for. Neither test is performed by the Intersect() function.

IParameters() Code

```
inline bool IParameters(//<==============PARAMETERS FOR LINE-PLANE INTERSECTION
    double t[3],//<------INTERSECTION PARAMETERS (CALCULATED BY THIS FUNCTION)
    const double T[9],//<--A TRIANGLE {T0X,T0Y,T0Z,T1X,T1Y,T1Z,T2X,T2Y,T2Z}
    const double S[6],//<---------------------A LINE {S0X,S0Y,S0Z,S1X,S1Y,S1Z}
    double e=1E-9){//<------CUTOFF VALUE FOR DETERMINING IF S & T ARE PARALLEL
  double A0=S[0]-S[3] , A1=T[3]-T[0] , A2=T[6]-T[0],//..............A in y=A*t
         A3=S[1]-S[4] , A4=T[4]-T[1] , A5=T[7]-T[1],
         A6=S[2]-S[5] , A7=T[5]-T[2] , A8=T[8]-T[2];
  double d=A0*A4*A8-A0*A5*A7 + A1*A5*A6-A1*A3*A8 + A2*A3*A7-A2*A4*A6;//..d=|A|
  if(fabs(d)<e)return false;//....=> S & T are parallel (and t is meaningless)
  double B0=(A4*A8-A5*A7)/d,B1=(A2*A7-A1*A8)/d,B2=(A1*A5-A2*A4)/d,//.A inverse
         B3=(A5*A6-A3*A8)/d,B4=(A0*A8-A2*A6)/d,B5=(A2*A3-A0*A5)/d,
         B6=(A3*A7-A4*A6)/d,B7=(A1*A6-A0*A7)/d,B8=(A0*A4-A1*A3)/d;
  double y0=S[0]-T[0],y1=S[1]-T[1],y2=S[2]-T[2];//..................y in y=A*t
  t[0]=B0*y0+B1*y1+B2*y2 , t[1]=B3*y0+B4*y1+B5*y2 , t[2]=B6*y0+B7*y1+B8*y2;
  return true;//.........................................=> S & T intersect
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
```

IParameters() Parameters

t **t** is a three-element array that stores the parameters described in equation 18. Note that **t** is modified by the IParameters() function.

T **T** is a nine-element array that stores the triangle that is defined by equation 13. $\mathbf{T}=\{T_{0,x},T_{0,y},T_{0,z},T_{1,x},T_{1,y},T_{1,z},T_{2,x},T_{2,y},T_{2,z}\}$.

S **S** is a six-element array that stores the line that is defined by equation 11. $\mathbf{S}=\{S_{0,x},S_{0,y},S_{0,z},S_{1,x},S_{1,y},S_{1,z}\}$.

e **e** is the cutoff value for testing whether or not **S** and **T** are parallel. If the determinant of the matrix in equation 18 is less than **e**, then **S** and **T** are considered to be parallel. The default value of **e** is 10^{-9}.

IParameters() Return Value

IParameters() returns false if **S** is parallel to **T**. A return value of false indicates that **t** has not been calculated and, thus, should not be passed to the Intersect() function.

Intersect() Code

```
inline bool Intersect(//<==============CALCULATES LINE-PLANE INTERSECTION POINT
    double x[3],//<--------POINT OF INTERSECTION (CALCULATED BY THIS FUNCTION)
    const double t[3],//<---------INTERSECTION PARAMETERS (FROM IParameters())
    const double S[6]){//<--------------------A LINE {S0X,S0Y,S0Z,S1X,S1Y,S1Z}
  for(int i=0;i<3;++i)x[i]=S[i]+(S[i+3]-S[i])*t[0];
  return t[1]>0&&t[1]<1&&t[2]>0&&t[2]<1;//.........S intersects parallelogram?
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
```

Intersect() Parameters

x **x** is a three-element array. Note that a purpose of the function is to calculate **x**.

t **t** is a parameter list that has been precalculated using the IParameters() function.

S **S** is a six-element array that stores the line that is defined by equation 11. $\mathbf{S}=\{S_{0,x},S_{0,y},S_{0,z},S_{1,x},S_{1,y},S_{1,z}\}$.

Intersect() Return Value

Intersect() returns true if line **S** intersects the parallelogram shown in figure 3. Note that additional checks can be performed to determine whether or not the point of intersection, **x** lies within the triangle defined by **T**[0], **T**[1], and **T**[2], or between the points on line **S**, **S**[0] and **S**[1]. See equations 20 and 21 for the additional checks.

Intersect() Example

Figure 4 shows the point, $\vec{p}$, where line-segment S intersects triangle T.

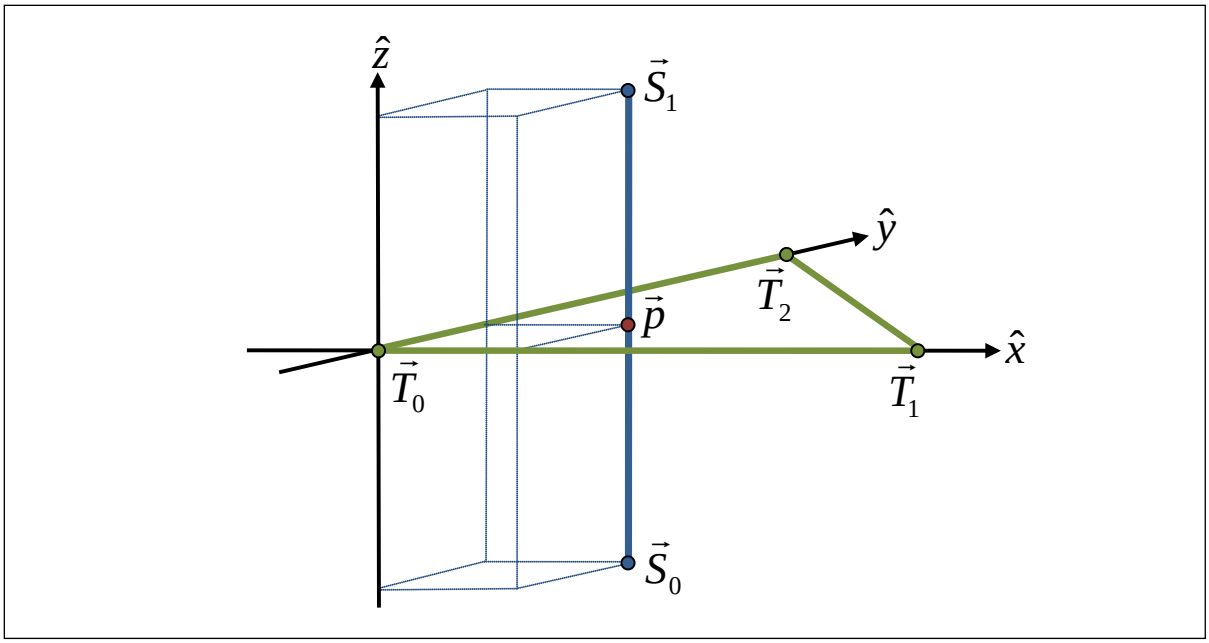


Figure 4. Intersect() example.

Let $\vec{S}_0 = \{1,1,-2\}$, $\vec{S}_1 = \{1,1,2\}$, $\vec{T}_0 = \{0,0,0\}$, $\vec{T}_1 = \{4,0,0\}$, and $\vec{T}_0 = \{0,4,0\}$. Point $\vec{p}$ can be found by first calling the IParameters() function, then using the result in the Intersect() function.

```
#include <cstdio>//.....................................................printf()
#include "y_3d_ops.h"//...............................IParameters(),Intersect()
int main(){
  double S[6]={1,1,-2,1,1,2};//.....................................a line segment
  double T[9]={0,0,0,4,0,0,0,4,0};//.................................a triangle
  double t[3];/*<-*/y3DOps::IParameters(t,T,S);//........intersection parameters
  double p[3];/*<-*/y3DOps::Intersect(p,t,S);//............point of intersection
  printf("p[0]=%f, p[1]=%f, p[2]=%f\n",p[0],p[1],p[2]);
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
```

OUTPUT:

```
p[0]=1.000000, p[1]=1.000000, p[2]=0.000000
```

5. Summary

A summary sheet is presented at the end of this report. It presents the y3DOps namespace, which contains the five functions that are described in detail in sections 2, 3, and 4 of this report. Also presented are two examples that demonstrate the versatility of the functions described in this report. The first uses the Rotate() function to calculate a set of points that defines the surface of a sphere. The second uses the Intersect() function to shade the blades of a pinwheel. Both functions create text files that contain all of the information needed to create the two images presented in the summary sheet. The actual images that are presented were created using the yGraphics namespace, which will be documented in a future report.

y3DOps Summary

y_3d_ops.h

```
#ifndef Y_3D_OPS_H_
#define Y_3D_OPS_H_
#include <cmath>//.........................................sin(),cos(),abs()
namespace y3DOps{//------ALL FUNCTIONS ASSUME RIGHT-HANDED CARTESIAN COORDINATES
  //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  inline void Translate(//<==============================PERFORMS A TRANSLATION
      double p[3],//<-------COORDINATES TO TRANSLATE (MODIFIED BY THIS FUNCTION)
      const double d[3]){//<---------------------------------DISPLACEMENT VECTOR
    p[0]+=d[0] , p[1]+=d[1] , p[2]+=d[2];
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
  inline void Rotate(//<====================================PERFORMS A ROTATION
      double p[3],//<----------COORDINATES TO ROTATE (MODIFIED BY THIS FUNCTION)
      const double o[3],//<-------------------THE ORIGIN OF THE AXIS OF ROTATION
      const double R[9]){//<------------------A ROTATION MATRIX (FROM RMatrix())
    double t0=p[0]-o[0],t1=p[1]-o[1],t2=p[2]-o[2];//.....translate to the origin
    p[0]=R[0]*t0+R[1]*t1+R[2]*t2+o[0];//...........................rotate, then
    p[1]=R[3]*t0+R[4]*t1+R[5]*t2+o[1];//                                 reverse
    p[2]=R[6]*t0+R[7]*t1+R[8]*t2+o[2];//                             translation
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
  inline void RMatrix(//<========================CALCULATES A 3D ROTATION MATRIX
      double R[9],//<--------------ROTATION MATRIX (CALCULATED BY THIS FUNCTION)
      const double v[3],//<-------THE AXIS OF ROTATION (V MUST BE A UNIT VECTOR)
      double rads){//<--THE ANGLE OF ROTATION (CCW ABOUT V, USE RIGHT-HAND RULE)
    double c=cos(rads),d=1-c,s=sin(rads),v0=v[0],v1=v[1],v2=v[2];
    R[0]=v0*v0*d+   c , R[1]=v0*v1*d-v2*s , R[2]=v0*v2*d+v1*s ;
    R[3]=v1*v0*d+v2*s , R[4]=v1*v1*d+   c , R[5]=v1*v2*d-v0*s ;
    R[6]=v2*v0*d-v1*s , R[7]=v2*v1*d+v0*s , R[8]=v2*v2*d+   c ;
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
  inline bool Intersect(//<============CALCULATES LINE-PLANE INTERSECTION POINT
      double x[3],//<--------POINT OF INTERSECTION (CALCULATED BY THIS FUNCTION)
      const double t[3],//<---------INTERSECTION PARAMETERS (FROM IParameters())
      const double S[6]){//<--------------------A LINE {S0X,S0Y,S0Z,S1X,S1Y,S1Z}
    for(int i=0;i<3;++i)x[i]=S[i]+(S[i+3]-S[i])*t[0];
    return t[1]>0&&t[1]<1&&t[2]>0&&t[2]<1;//.........S intersects parallelogram?
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
  inline bool IParameters(//<============PARAMETERS FOR LINE-PLANE INTERSECTION
      double t[3],//<------INTERSECTION PARAMETERS (CALCULATED BY THIS FUNCTION)
      const double T[9],//<--A TRIANGLE {T0X,T0Y,T0Z,T1X,T1Y,T1Z,T2X,T2Y,T2Z}
      const double S[6],//<--------------------A LINE {S0X,S0Y,S0Z,S1X,S1Y,S1Z}
      double e=1E-9){//<------CUTOFF VALUE FOR DETERMINING IF S & T ARE PARALLEL
    double A0=S[0]-S[3] , A1=T[3]-T[0] , A2=T[6]-T[0],//.............A in y=A*t
           A3=S[1]-S[4] , A4=T[4]-T[1] , A5=T[7]-T[1],
           A6=S[2]-S[5] , A7=T[5]-T[2] , A8=T[8]-T[2];
    double d=A0*A4*A8-A0*A5*A7 + A1*A5*A6-A1*A3*A8 + A2*A3*A7-A2*A4*A6;//..d=|A|
    if(fabs(d)<e)return false;//....=> S & T are parallel (and t is meaningless)
    double B0=(A4*A8-A5*A7)/d,B1=(A2*A7-A1*A8)/d,B2=(A1*A5-A2*A4)/d,//.A inverse
           B3=(A5*A6-A3*A8)/d,B4=(A0*A8-A2*A6)/d,B5=(A2*A3-A0*A5)/d,
           B6=(A3*A7-A4*A6)/d,B7=(A1*A6-A0*A7)/d,B8=(A0*A4-A1*A3)/d;
    double y0=S[0]-T[0],y1=S[1]-T[1],y2=S[2]-T[2];//.................y in y=A*t
    t[0]=B0*y0+B1*y1+B2*y2 , t[1]=B3*y0+B4*y1+B5*y2 , t[2]=B6*y0+B7*y1+B8*y2;
    return true;//....................................=> S & T intersect
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
}//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
#endif
```

IMAGE 1

image created from sphere.txt.

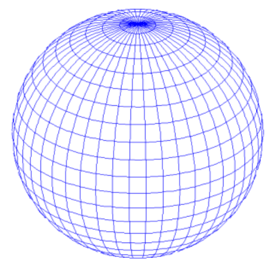
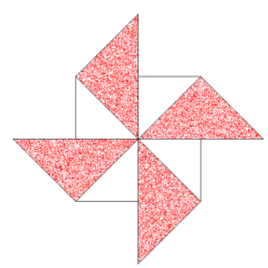


IMAGE 2

image created from pinwheel.txt.



EXAMPLE 1

```
#include <algorithm>//...................................................copy()
#include <cstdio>//.....................FILE,freopen(),stdout,printf(),fclose()
#include "y_3d_ops.h"//.....................................RMatrix(),Rotate()
int main(){//<===================================USE ROTATIONS TO DRAW A SPHERE
  using namespace y3DOps;
  using std::copy;
  double x[3]={1,0,0} , y[3]={0,1,0} , z[3]={0,0,1} , o[3]={0,0,0},a[3]={0,1,0};
  double H[200][23][3]={0,0,1} , V[200][23][3]={1,0,0};//.......lat & long lines
  double Ry[9];/*<-*/RMatrix(Ry,y,3.14159/(23+1));
  double Rz[9];/*<-*/RMatrix(Rz,z,2*3.14159/(200-1));
  double R3[9];/*<-*/RMatrix(R3,z,3.14159/(23));
  for(int j=0;j<23;++j){
    if(j!=0) Rotate(copy(H[0][j-1],H[0][j-1]+3,H[0][j])-3,o,Ry);
    for(int i=1;i<200;++i)Rotate(copy(H[i-1][j],H[i-1][j]+3,H[i][j])-3,o,Rz);
    if(j!=0) Rotate(copy(V[0][j-1],V[0][j-1]+3,V[0][j])-3,o,R3),Rotate(a,o,R3);
    double Ra[9];/*<*/RMatrix(Ra,a,2*3.14159/(200-1));
    for(int i=1;i<200;++i)Rotate(copy(V[i-1][j],V[i-1][j]+3,V[i][j])-3,o,Ra);}
  double R[9];/*<-*/RMatrix(R,x,-.5);
  for(int i=0,j;i<200;++i)for(j=0;j<23;++j)//tilt top of sphere towards observer
    Rotate(H[i][j],o,R),Rotate(V[i][j],o,R);
  FILE *f=freopen("sphere.txt","w",stdout);//.........redirect output to a file
  for(int i=0;i<200;++i)for(int j=0;j<23;++j){
    for(int k=0;k<3;++k)printf("%6.3f,",H[i][j][k]);
    for(int k=0;k<3;++k)printf("%6.3f%s",V[i][j][k],j==23-1&&k==2?"\n":",");}
  fclose(f);
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
```

EXAMPLE 2

```
#include <cstdio>//.....................FILE,freopen(),stdout,printf(),fclose()
#include <cstdlib>//...........................................rand(),RAND_MAX
#include "y_3d_ops.h"//............RMatrix(),Rotate(),IParameters(),Intersect()
int main(){//<================USE ROTATIONS AND INTERSECTIONS TO DRAW A PINWHEEL
  using namespace y3DOps;
  double T[4][9]={0,0,0 , 0,1,0 , -.5,.5,0};
  double T2[4][9]={0,0,0 , -.5,.5,0 , -.5,0,0};
  const double z[3]={0,0,1} , o[3]={0,0,0};
  double R[9];/*<-*/RMatrix(R,z,3.14159/2);
  for(int i=1;i<4;++i)for(int j=0;j<3;++j){
    for(int k=0;k<3;++k)T2[i][3*j+k]=T2[i-1][3*j+k];/*->*/Rotate(T2[i]+3*j,o,R);
    for(int k=0;k<3;++k)T[i][3*j+k]=T[i-1][3*j+k];/*->*/Rotate(T[i]+3*j,o,R);}
  FILE *f=freopen("pinwheel.txt","w",stdout);//........redirect output to a file
  for(int i=0;i<100000;++i){
    double S[6]={rand()*2./RAND_MAX-1,rand()*2./RAND_MAX-1,
      1,S[0],S[1],-1};
    for(int k=0;k<4;++k){
      double t[3];/*<-*/IParameters(t,T[k],S);
      double x[3];
      if(Intersect(x,t,S)&&t[1]+t[2]<1)
        printf("%6.3f,%6.3f,%6.3f\n",x[0],x[1],x[2]);}}
  fclose(f);
  FILE *g=freopen("outline.txt","w",stdout);//.........redirect output to a file
  for(int i=0;i<4;++i)for(int j=0;j<3;++j)for(int k=0;k<3;++k)
    printf("%6.3f%s",T[i][3*j+k],k==2?"\n":",");
  for(int i=0;i<4;++i)for(int j=0;j<3;++j)for(int k=0;k<3;++k)
    printf("%6.3f%s",T2[i][3*j+k],k==2?"\n":",");
  fclose(g);
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~02MAY2013~~~~~~
```

NO. OF
COPIES ORGANIZATION

1 (PDF) DEFENSE TECHNICAL INFORMATION CTR DTIC OCA

1 (PDF) DIRECTOR US ARMY RESEARCH LAB IMAL HRA

1 (PDF) DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL

1 (PDF) GOVT PRINTG OFC A MALHOTRA

ABERDEEN PROVING GROUND

1 (PDF) DIR USARL RDRL WML A R YAGER

INTENTIONALLY LEFT BLANK.