

# High Dimensional Rendering in OpenGL

Josh McCoy

December 1, 2003)

## Description of Project

Adding high dimensional rendering capability to the OpenGL graphics programming environment is the goal of this project. We will approach this in three steps. First we will generalize high dimensional mathematics from algorithms used in lower dimensional graphics theory and OpenGL itself. The high dimensional mathematics will then be implemented as a wrapper to OpenGL. Finally, the project will be announced as an open source project and made publicly available.

## Why This Project is Important

### What is OpenGL?

In 1992, SGI started an initiative to create a consolidated and standard application programming interface (API) that would be available to all vendors for the development of graphics applications. The result was the OpenGL API based on the IRIS GL<sup>TM</sup> library. SGI then released a sample implementation of OpenGL under an open source license to help other vendors who considering implementing OpenGL themselves. The OpenGL Architecture Review Board was then established with the power to control and modify the OpenGL API [5].

OpenGL has two public faces. One is a cross-platform runtime library that provides 3d hardware acceleration and 3d rendering. It is readily available and is included with the Mac OS, Windows, and Linux operating systems [3]. The other face is displayed to industry and developers. OpenGL is

presented as an environment for developing 2d and 3d graphics applications. The OpenGL API has become industry's most pervasive 2d and 3d graphics API. Thousands of applications have been developed on many platforms using the OpenGL API. A broad set of rendering, texture mapping, and other useful visualization functions are supported by OpenGL [4]. In the context of this project, we will look at OpenGL's developer-oriented face.

## Potential Uses

Adding high dimensional capability to a popular and pervasive environment for developing interactive graphics seems to be a worthwhile endeavor. As OpenGL is popular enough to be considered an industry standard [4], adding any additional functionality has the potential to benefit many people and projects. Given the complexity of data that is currently being mined and analyzed in both industry and academia, adding a way to visualize complex information to OpenGL could and probably will be of value to many.

“The difficult task of sifting through the data generated by the extensive use of computers today—and locating concise and interpretable information within that data—is called knowledge discovery databases (KDD)” [6]. Visualizing this high dimensional data is an important step to KDD paradigm [6]. Paired with an a high dimensional-capable version of OpenGL, a high dimensional data set can be visualized using a powerful, well known, open source tool.

Applications in industry that deal with higher dimensions could benefit from an easy method of visualizing complex data sets. A potent example of high dimensional data can be found in the car insurance business. Car insurance companies use collected data from accidents, expenses, and consumers to estimate insurance premia (Chapados 1-2). High dimensional data inference in tandem with visual rendering of the information could benefit both the company and the consumer.

## Mathematics of High Dimensional Graphics

In order to understand how graphics are rendered beyond the third dimension, general understanding of graphics in the second and third dimensions is necessary. Linear algebra provides a foundation for the basic algorithms and concepts used in rendering viewable graphics. This knowledge can be

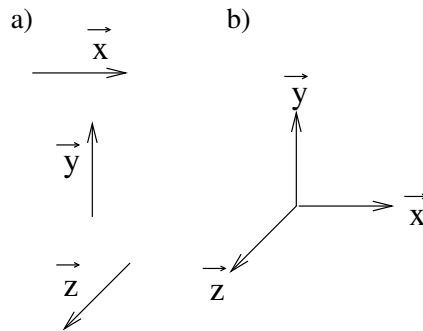


Figure 1: Unit vectors and unit coordinate system. a) unit vectors b) unit vectors combined to create a coordinate system.

expanded to work in high dimensions through generalization. The generalized concepts will be the basis for adding high dimensional capability to OpenGL.

## The Basis: Lower Dimensional Graphics

### Homogeneous Coordinate System

A coordinate system made of unit vectors describes a space in which points and vectors can be plotted (Figure 1).

In the unit space, points are a location and vectors are both magnitude and direction. They are represented by an  $n+1$ -tuple of values corresponding to the dimensionality of the space. An example of a point represented in two dimensions is the 3-tuple,  $\langle 1, 2, 1 \rangle$ . The 5-tuple  $\langle 3, 4, 5, 0 \rangle$  represents a vector in three dimensional space. In a point, the values of the first  $n$  elements of the  $n+1$ -tuple is a scalar that describe a relationship to a unit vector. When a tuple is considered a vector, the scalars represent magnitude in relation to the coordinate system. These scalar relations to the unit vectors determine how the point or vector will be represented in the unit space. For example, the point  $\langle 1, 2, 3 \rangle$  can also be thought of as  $1\vec{x} + 2\vec{y} + 3\vec{z}$  from the point  $\langle 0, 0, 0, 1 \rangle$ . (Figure 2).

As the representation of a point and a vector are similar, homogeneous-coordinate representations are used in computer graphics[1]. A homogeneous-coordinate representation of a point and a vector differ in the the value  $n+1$  element of the tuple,  $P$ . The new element has a value of 1 if the tuple represents a point or a value of 0 if it defines a vector. Any value other than

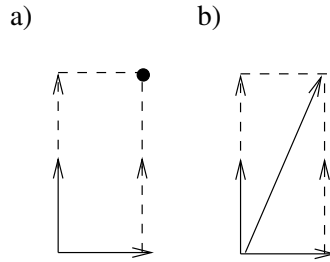


Figure 2: Vectors and points in a unit coordinate system. a) Point b) Vector

1 or 0 for  $P$  is invalid. The elegance of the choice of values for  $P$  is revealed when points and vectors are involved in mathematical operations. When a point,  $\langle x_1, y_1, z_1, 1 \rangle$ , is added to a vector,  $\langle x_2, y_2, z_2, 0 \rangle$ , the mathematical and intuitive results agree that the result is the point  $\langle x_1 + x_2, y_1 + y_2, z_1 + z_2, 1 \rangle$ . Two vectors ( $\langle x_1, y_1, z_1, 0 \rangle$  and  $\langle x_2, y_2, z_2, 0 \rangle$ ) added together result in the vector  $\langle x_1 + x_2, y_1 + y_2, z_1 + z_2, 0 \rangle$ . The addition of two points has no intuitive result. This is mirrored by the invalid sum ( $P = 2$ ).

### Translation and Rotation

A standard way of manipulating computer graphics is through the use of transformation matrices [1]. For this to work, a homogeneous coordinate tuple must be thought of as either a  $1 \times n + 1$  or a  $n + 1 \times 1$  matrix. The new homogeneous coordinate matrix is multiplied by a transformation matrix and results in the homogeneous coordinate being changed and moved to fit the transformed space<sup>1</sup>. Two common and necessary transformation matrices used in rendering objects are the translation and rotation matrices.

A translation matrix is used for displacing objects with respect to the origin of the coordinate system. The object is moved from its initial position to a new position. Since the displacement has magnitude and direction, it can be conceptualized as a vector. A point,  $p$  to be displaced by a translation matrix,  $T$ , results in a new point  $Tp = p_T$ . Given a displacement vector,  $\langle x_d, y_d, z_d, 1 \rangle$ , the transformation matrix responsible for the translation can be found in Figure 3.

Rotation transformations need a vector to rotate around. A simple subset of rotations are those that rotate around one of the coordinate system's unit

---

<sup>1</sup>If the transformation matrix is the identity matrix the homogeneous coordinates will not change.

$$Tp = p_T$$

$$\begin{pmatrix} 1 & 0 & 0 & x_d \\ 0 & 1 & 0 & y_d \\ 0 & 0 & 1 & z_d \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x + x_d \\ y + y_d \\ z + z_d \\ 1 \end{pmatrix}$$

Figure 3: Translation transformation.

$$R_x p = p_R$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \cos \theta - z \sin \theta \\ y \sin \theta + z \cos \theta \\ 1 \end{pmatrix}$$

Figure 4: Rotation transformation.

vectors. These rotations can be thought of as positive rotations around an axis by some angle,  $\theta$ . The equation is  $R_a p = p_R$  where  $R_a$  is the transformation matrix for rotating around an arbitrary axis,  $p$  is the original point, and  $p_R$  is the transformed point. The values of the homogeneous coordinates that correspond to the axis of rotation do not change but the values in the other two dimensions change. An example is a  $\theta$  degree rotation around the x axis (Figure 4).

The unit vectors (axes) are not the only vectors around which an object can be rotated. Rotation can occur around arbitrary vectors. The magnitude of change between two points,  $p_1$  and  $p_2$ , is used to specify the vector to be rotated around. The center, fixed point does not need to be the origin and is now any point,  $p_0$ . The last parameter is the angle of rotation,  $\theta$ .

“According to Euler’s rotation theorem, any rotation may be described using three angles” [7]. In order to rotate around an arbitrary vector,  $\vec{r}$ , the angles the vector makes with the axes ( $\theta_x$ ,  $\theta_y$ , and  $\theta_z$ ) must be found. An example would be finding  $\theta_y$ . Two vectors are needed to determine the value of the angle. The first vector,  $\vec{r}_y$ , is the arbitrary vector projected into 2d by ignoring the dimension along the y-axis. This vector lies on the plan created by the plane created by the axes of the next two dimensions or the z and x

$$a) \begin{pmatrix} x \\ 0 \\ z \\ 0 \end{pmatrix} \quad b) \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \quad c) \vec{r}_y \cdot \vec{z} = |\vec{r}_y| |\vec{z}| \cos \theta_y \quad d) \cos^{-1} \left( \frac{\vec{r}_y \cdot \vec{z}}{|\vec{r}_y| |\vec{z}|} \right) = \theta_y$$

Figure 5: Finding the angles to axes from an arbitrary vector. a)  $\vec{r}_y$  b)  $\vec{z}$  c) dot product equation d) equation to find  $\theta_y$

axes. The second vector needed is the unit vector corresponding to the axis of the  $n+1$  dimension,  $\vec{z}$  (Figure 5).

An angle from  $\vec{r}$  to the x-axis can be found by taking the dot product of the vector where the component corresponding to the x-axis is 0 and the unit vector of the plane perpendicular to the axis. The process is repeated to find  $\theta_y$  and  $\theta_z$ .

To further simplify the problem, translating the fixed point to the axis allows the rotations to be simply done [8]. To actually compute the rotation, one must translate  $P_0$  to the origin, rotate around  $\theta_x$ ,  $\theta_y$ , and  $\theta_z$ , and translate back to  $P_0$ .

### Orthographic Projection

In rendering and visualization, a projection reproduces an image in a different way. Projections are used to visualize higher dimensional images to lower dimensions. OpenGL displays rendered 3d images to a 2d screen through projection. A simple type of projection is the orthographic projection. By simply ignoring the highest dimension, orthographic projections present a view with no perspective (Figure 6).

Transformation matrices are used when projecting orthographically as they are used when translating and rotating. The 3d orthographic projection matrix preserves the values of the first two dimensions and truncates the values of the third dimension (Figure 6).

### Perspective Projection

Perspective projection results in a visualization similar to what human perceive. Unlike orthographic projections, perspective projections only alter the

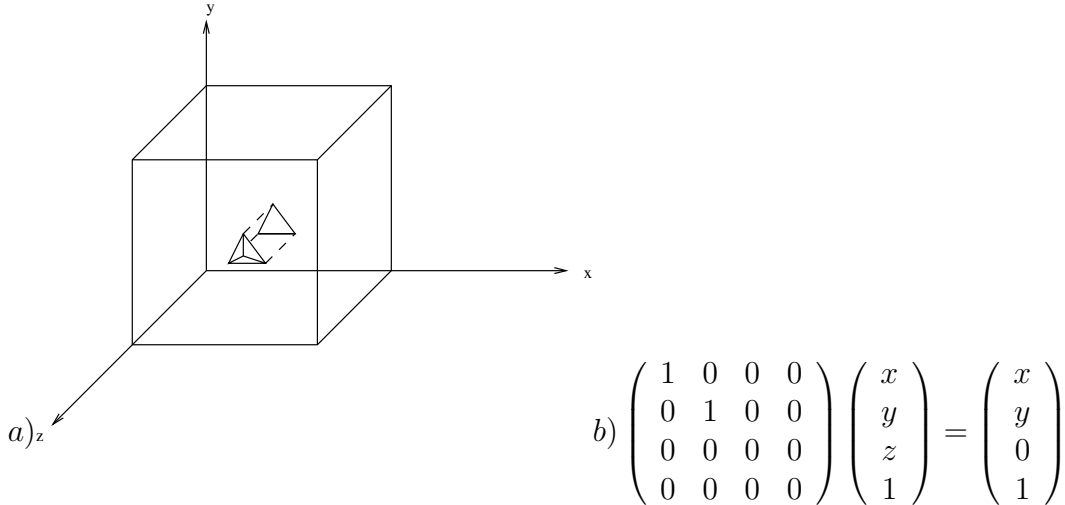


Figure 6: 3d to 2d orthographic projection and transformation matrix.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z/d \end{pmatrix} \Rightarrow \begin{pmatrix} \frac{x \cdot d}{z} \\ \frac{y \cdot d}{z} \\ z \\ d \end{pmatrix}$$

Figure 7: 3d to 2d perspective projection matrix and perspective division.

data of the highest dimension. The points and vectors altered are moved to the same place in that dimension. In 3d perspective projections, the 3d points are projected to a plane given by a location on the z-axis.

Perspective projections require a value in the highest dimension,  $d$ , to project to. The inverse of  $d$  is in the  $n+1$  row,  $n$ th column element of a matrix similar to the identity matrix with the last element in the diagonal having a value of 0 (Figure 7). A homogeneous coordinate altered by a perspective projection matrix in which  $d \neq 1$  will have a value for  $P$  that is not 0 or 1. For the coordinate to make sense, perspective division must be performed. Dividing all the elements of a homogeneous coordinate by the value of  $P$  is perspective division (Figure 7).

$$Tp = p_T$$

$$\begin{pmatrix} 1 & 0 & \dots & x_{d1} \\ \vdots & \ddots & \vdots & \vdots \\ \vdots & \dots & \ddots & x_{dn} \\ 0 & \dots & \dots & 1 \end{pmatrix} \begin{pmatrix} x_{d1} \\ \vdots \\ x_{dn} \\ 1 \end{pmatrix} = \begin{pmatrix} x + x_{d1} \\ \vdots \\ x_n + x_{dn} \\ 1 \end{pmatrix}$$

Figure 8: Translation transformation.

## The Next Step: Higher Dimensional Graphics

With the mathematics of low dimensional visualization as a foundation, algorithms for high dimensional visualization can be generalized. The following algorithms are the basis for adding high dimensional rendering capability to OpenGL.

### Homogeneous Coordinate System

The generalization of homogeneous coordinates is straight forward. They are  $n+1$ -tuples of numbers where  $n$  is the number of dimensions the coordinate will represent. Similar to low dimension homogeneous coordinates, the first  $n$  elements of the tuple are scalar values of the unit vectors. An example is the five dimensional vector,  $\langle 3, 6, 9, 2, 4, 0 \rangle$ . The last value is the  $P$  value. The generalization is:

$$\langle x_1, x_2, \dots, x_n, P \rangle$$

### Translation and Rotation

Translation matrices are generalized by a  $n+1 \times n+1$  matrix,  $T$ . The vector representing the displacement of the translation is a  $n+1$ -tuple homogeneous coordinate (Figure 8).

Rotations around the unit vectors in  $n$  dimensions is similar to that of lower dimensions. The rotation matrix,  $R_a$ , rotates around the unit vector that corresponds with the  $a$ th axis. Both the  $a+1$  and the  $a+2$  elements of any homogeneous vector transformed by  $R_a$  are changed to reflect the rotation (Figure 9).

$$R_a p = p_R$$

$$\begin{pmatrix} 1 & 0 & \dots & \dots & \dots & \dots & 0 \\ \vdots & \ddots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \dots & \cos \theta & -\sin \theta & \ddots & \ddots & \vdots \\ \vdots & \dots & \sin \theta & \cos \theta & \ddots & \ddots & \vdots \\ \vdots & \dots & \ddots & \dots & 1 & \ddots & \vdots \\ \vdots & \dots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & \dots & \dots & \dots & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ x_n \\ 1 \end{pmatrix} = \begin{pmatrix} x_1 \\ \vdots \\ x_{a+1} \cos \theta - x_{a+1} \sin \theta \\ x_{a+2} \sin \theta + x_{a+2} \cos \theta \\ \vdots \\ x_n \\ 1 \end{pmatrix}$$

Figure 9: Generalized rotation transformation.

### Rotation Around an Arbitrary Vector

By using vector arithmetic, rotating around an arbitrary vector in n dimensions is very similar to the algorithm in three dimensions. The vector to be rotated around is considered normal to a plane of rotation. By taking the result of the dot product of the normal and a vector lying on an axis for each axis, the Euler angles required to rotate arbitrarily are found. Each angle is used to rotate around its corresponding axis resulting in a rotation around the arbitrary vector.

### Orthographic Projection

As orthographic projections simply ignore the highest dimension, they are readily adaptable to the n dimensional case. The translation matrix resembles a  $n + 1 \times n + 1$  identity matrix with the element in the nth row and nth column having a value of zero instead of one (Figure 10).

### Perspective Projection

In 3d, perspective projections are taken by placing all 3d points on a 2d space within the 3d space. Likewise, an n dimensional space can be projected with perspective on a  $n - 1$  dimensional space with the same method. A single point on the  $n^{th}$  axis,  $d$ , is the location where the n dimensional points are projected. This  $d$  value is in the perspective projection translation matrix inverted as the element in the  $n + 1$  row and the  $n$  column (Figure 10).

$$\begin{pmatrix} 1 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 & \vdots \\ 0 & \cdots & \cdots & \frac{1}{d} & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \\ 1 \end{pmatrix} = \begin{pmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \\ \frac{x_n}{d} \end{pmatrix} \Rightarrow \begin{pmatrix} \frac{x_1 \cdot d}{x_n} \\ \vdots \\ \vdots \\ \frac{x_{n-1} \cdot d}{x_n} \\ d \\ 1 \end{pmatrix}$$

Figure 10: Generalized Projection Matrices

The product of a homogeneous coordinate and the perspective projection matrix need to have perspective division performed on it just as in the 3d case. Perspective division results in a homogeneous coordinate with the value corresponding to the  $n$ th dimension being  $d$ .

When considered with an  $n$  dimensional viewing volume, perspective projection can make a  $n$  dimensional space into any lower dimensional space. Of particular interest is the  $n$  to 3 dimensions case. This makes higher dimensional objects viewable in OpenGL. By using projection matrices of decreasing dimensions, perspective division, and an  $n$  dimensional viewing volume<sup>2</sup>, 3d projections of higher dimensional spaces can be realized.

## Putting It All Together: OpenGL Pipeline

OpenGL combines both state machine and pipeline approaches when rendering. A few examples the state variables are point size, current background color, drawing color, and lighting options. They persist until changed. The pipeline is the order in which OpenGL does various rendering tasks. Performing transformation (setting the modelview and projection matrices), clipping, projection, and rasterization in order is the generalized pipeline. The pipeline process and states variables are codependent.

---

<sup>2</sup>The “near” face of the viewing volume in each dimension defines a  $d$  value for each dimension.

## Wrapping

Two major factors in the creation of the nd wrapper for OpenGL were to imbue it with the functionality of rendering higher dimensional objects in 3d and to keep the the feel and functionality of the wrapper's interface as close to those of OpenGL as possible. In keeping the feel of OpenGL, the wrapper has an internal state machine. The rendering process is also pipelined. The function call sequence to render an object follows OpenGL's syntax. OpenGL begins and ends rendering with `glBegin(...)` and `glEnd()` function calls. The wrapper uses `glBeginNd(...)` and `glEndNd()` function calls. Specifying vertices's in OpenGL is done with calls similar to `glVertex3df(...)` while the wrapper uses `glVertexNdf(...)`. The function naming convention is kept when dealing with the modelview and projection matrices. An example would be rotating the modelview matrix id done with a call to `glRotate(...)` in OpenGL and with `glRotateNd(...)` in the wrapper.

The wrapper's internal state machine stores the information needed to render the n dimensional objects in 3d with a set of internal classes. A class stores the dimension to be rendered, the n dimensional modelview and projection matrices, and the n dimensional viewing volume. Another set of classes is used to store the call information (such as the location of a vertex, drawing mode, current drawing color, etc). When the n dimensional calls are complete, the wrapper projects the stored vertices to 3d and makes 3d OpenGL calls using the stored call information.

## Summary and Future Work

Although the basics for rendering graphics in higher dimensions exist within the wrapper, there are still a few tasks that remain to be done. The next most important task is the implementation of clipping in higher dimensions. As the wrapper functions now, clipping is done in the third dimension by OpenGL's clipping routines. This is inadequate when rendering higher dimensional data because the clipping after the higher dimensional image is rendered results in an inaccurate image. As Figure 11 illustrates in three dimensions, clipping after performing the projection transformation results in a image that is different than if the clipping was performed first in the same circumstances.

After the clipping problem is solved, the wrapper will be packaged and made publicly available and open sourced. An interactive demo based around

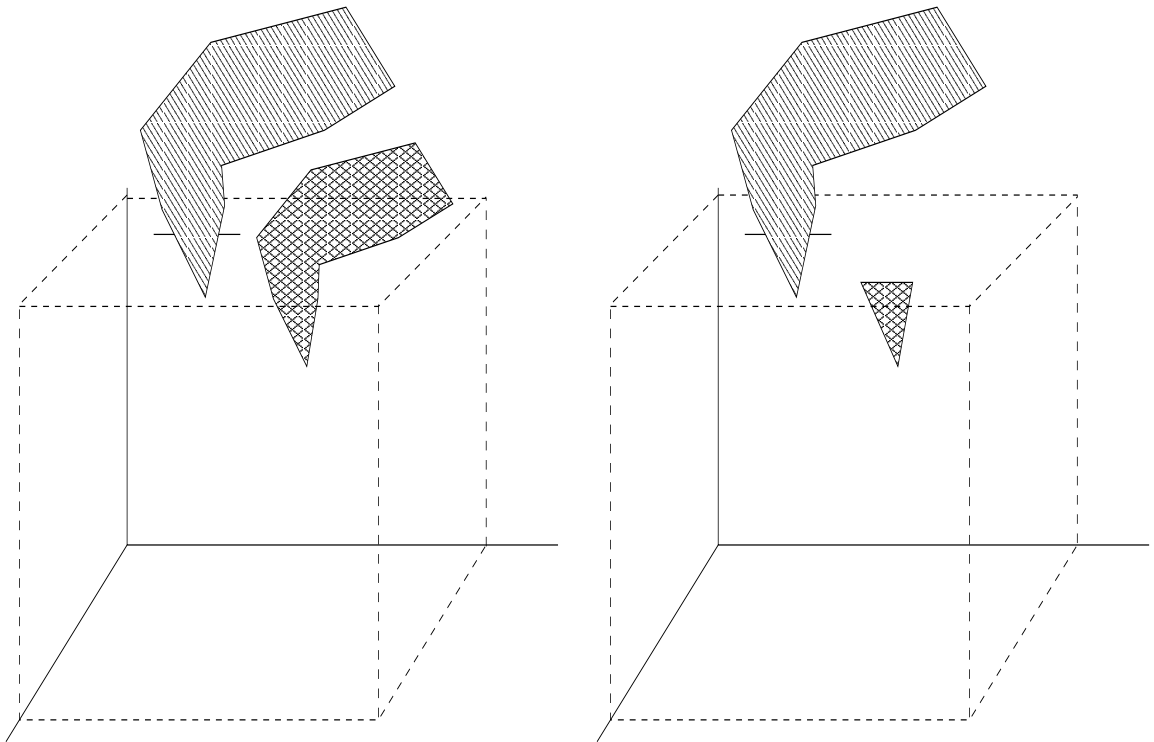


Figure 11: Clipping before and after projection.

an algorithm to draw an n dimensional hypercube will be bundled with the wrapper in order to show the wrapper's capability. Past this point, work on this projected will be determined by public response.

## References

- [1] Angel, Edward. *Interactive Computer Graphics: A Top-down Approach with OpenGL. 2nd ed.* Reading, Massachusetts: Addison Wesley Longman, Inc. 2000.
- [2] Jones, Christopher V. "Projection Approaches." *Interactive Transactions of ORMS*. September 1998. September 2003 <<http://catt.bus.okstate.edu/jones98/projecti.htm>>.
- [3] OpenGL Architecture Review Board. "Frequently Asked Questions for the Professional User and Gamer." *OpenGL*. 6 October 2003. <<http://www.opengl.org/users/faq/index.html>>.
- [4] OpenGL Architecture Review Board. "OpenGL Overview." *OpenGL*. 6 October 2003. <<http://www.opengl.org/developers/about/overview.html>>
- [5] Silicon Graphics, Inc. "SGI-OpenGL: Home Page." *Silicon Graphics, Inc.* 6 October 2003. <<http://www.sgi.com/software/opengl/>>.
- [6] Hinneburg, Alexander, Daniel Keim, and Markus Wawryniuk. "Using Projections to Visually Cluster High-Dimensional Data." *Computing in Science and Engineering* Vol. 5, Num. 2, pp. 15-25, March/April 2003.
- [7] Weisstein, Eric W. "Euler Angles." *Eric Weisstein's world of Mathematics*. September 2003 <<http://mathworld.wolfram.com/EulerAngles.html>>.
- [8] Hill, Francis S. Jr. *Computer Graphics Using Open GL Second Edition*. Prentice Hall,