3D Graphics Kit (Zraphics)

Khalid Al-Zamil
913052

Submitted to
Dr. Muhammad Sarfraz

Department of Information and Computer Science
College of Computer Science and Engineering
King Fahad University of Petroleum and Minerals
Dhahran - Kingdom of Saudi Arabia

2003

Abstract

Zraphics is a 3D graphics kit for the creation and manipulation of 2D and 3D shapes. Several algorithms have been developed to enrich Zraphics with the functionality that is provided by any moderate graphics package. The programming language and design issues used for the implementation of Zraphics are discussed. The description of the graphical user interface (GUI) along with the functionalities it provides are also mentioned. A discussion on how the shapes are generated along with their algorithms is also covered. In addition, the mechanism used to color fill shapes is discussed thoroughly. The transformation operations implemented are also mentioned in detail. Two viewing projections have been developed and are listed in the report. The painter's algorithm for hidden surface detection has been implemented as well as an algorithm to detect hidden lines. Finally, some light will be shed on how color shading is achieved in Zraphics along with its implementation.

1. Introduction

Computer graphics is a very important and interesting field. It is almost used in all disciplines such as the military, medicine and engineering. It simplifies the designing process that is needed in the industry (e.g. the design of cars, ships, robots … etc.). With the aid of computer graphics, engineers are capable of generating accurate and complex graphicss in a very short period of time, compared to the old fashioned way where they needed to do it manually.

It is true that computer graphics saves a lot in terms of the time needed to generate a specific design, not to mention the perfect accuracy, but the development of such a graphics package is not at all that trivial! Lots of mathematics, computer programming and plenty of patience is required in order to develop a sophisticated computer graphics package. This report is about the development of a 3D graphics kit called Zraphics (**Z**amil's G**raphics**). The package is capable of creating and rendering several 2D and 3D shapes, it also allows manipulating these shapes by applying certain transformation operations, in addition to providing the capability to maneuver through the 3D scene.

The shapes supported by Zraphics are: Lines, polygons, circles, ellipses, B-Spline curves, B-Spline surfaces, polyhedrons and spheres. The shapes can be displayed as wireframes or filled with a specific colour. Several transformation operations are also available namely: translation, scaling, rotation, shearing and reflection. Two viewing projections have also been developed to give a 3D effect when displaying the shapes. These viewing projections are the orthographic parallel projection and the perspective projection. Zraphics also provides the capability to detect both hidden surfaces and hidden lines. Colour shading is also implemented to give that live 3D effect.

This report is organized as follows; first the programming language and design issues used for the implementation are discussed in section 2. This is followed by a complete description of the graphical user interface (GUI) and the functionalities it provides (section 3). Section 4 is dedicated to a discussion on how the shapes are computed along with their algorithms. The mechanism used to color fill shapes is discussed thoroughly in section 5. The transformation operations implemented are mentioned in section 6. The viewing projections are covered in section 7. Section 8 covers the hidden surface and hidden line detection algorithms. Finally, section 9 will shed some light on color shading. There are plenty of images to illustrate the various concepts covered in this project, so buckle up and enjoy the ride!

2. Design Issues

Zraphics has been entirely developed using Java (version 1.4.1). Java has been selected as the programming language due to many advantages it has to offer, which include but not limited to:

1. Compatibility across platforms.
2. Object orientation.
3. Availability of many Integrated Development Environment Applications (IDEA) that boost the development time. The IDEA used in this project was IntelliJ.
4. Easier maintenance and future enhancements.

The design of Zraphics has been influenced by many object oriented design patterns. These design patterns allow for simpler code, easier maintenance, code reusability and future enhancements. Some of the different design patterns implemented will be mentioned along with the classes that take advantage of them.

To give an example of the design patterns implemented, 3 classes will be discussed namely: *Shape*, *Transformer* and *Viewer*. These are 3 out of a total of 44 classes developed. The *Shape* class is the top class for all possible shapes. It contains all members that are shared among all other shapes such as the world coordinates; it also includes common methods such as setting the screen coordinates. The *Shape* class is inherited in all the shape classes like *Sphere* and *Line*. Likewise, the *Transformer* Class is inherited in the *Translator*, *Rotator*, *Scaler*, *Shearer* and *Reflector* classes. The *Viewer* class is an example of a server class. It provides the displaying functionality and allows any interested class to register itself as a *listener* to receive a notification of any event that might occur. The listener design pattern prevents the code of becoming messy and tangled. As an example, if a *Transformer* object, say the *Translator*, is interested in receiving events from the *Viewer*, then all it has to do is register itself as a listener. Upon registration, the *Translator* will be notified about any events that happen, such as the creation or deletion of a shape.

3. The Graphical User Interface

The GUI has been developed with one objective in mind. This objective is to have Zraphics as intuitive and as user friendly as possible. In other words, to simplify the process of creating and modifying shapes, maneuvering within the 3D scene, and to apply different transformations and viewing projections. Figure 1 shows how Zraphics looks like. In the following sections, a description of the functionality provided by Zraphics is given.

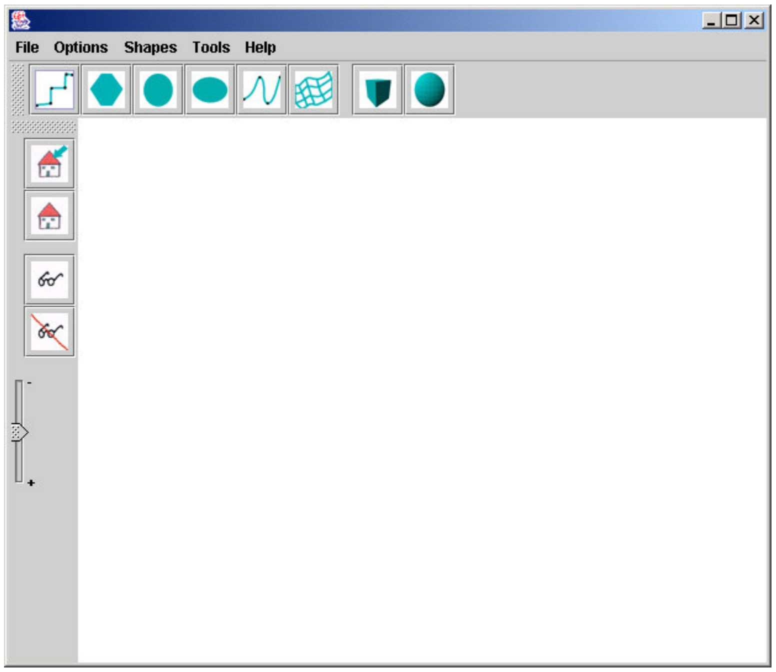


Figure 1: Zraphics main window.

3.1. Options

The Options button in the menu bar contains options for:

1. Setting the home view.
2. Going to the home view.
3. Setting the viewing projection to parallel projection.
4. Setting the viewing projection to perspective projection.
5. Showing the window options tool for displaying the axis and changing the background color (to be discussed in more detail in section 3.1.1).
6. Displaying the triangulations for a filled shape to show the triangulations used for Hidden Surface Detection (to be discussed further in section 8).

Upon clicking on the Options button a pull down menu will appear similar to figure 2.

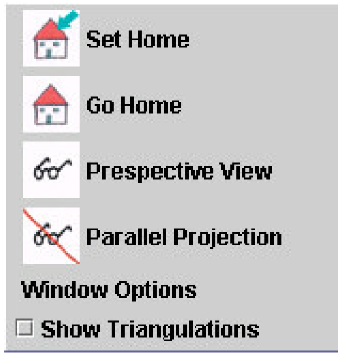


Figure 2: Options available in Zraphics.

3.1.1. Window Options

This is used to allow for changing the background for the 3D display. It also enables users to show the axis and control both the color and length for each axis (see figure 3).

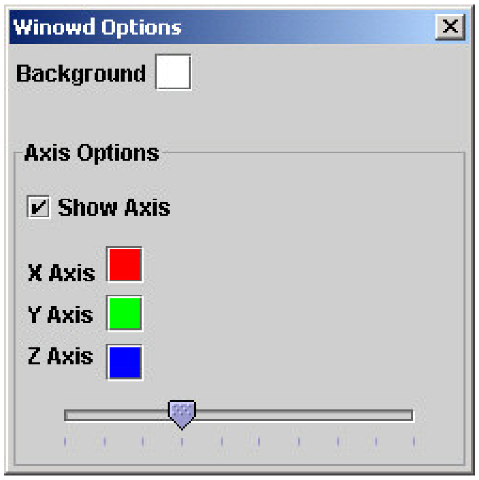


Figure 3: The window used for controlling the 3D scene background color and displaying the axis

3.2. Shapes

This button enables the creation of 2D and 3D shapes. The shapes supported by Zraphics are: Lines, polygons, circles, ellipses, B-Spline curves, B-Spline surfaces, polyhedrons and spheres. These are easily accessed by clicking on the Shapes button in the menu bar and will look something like figure 4.

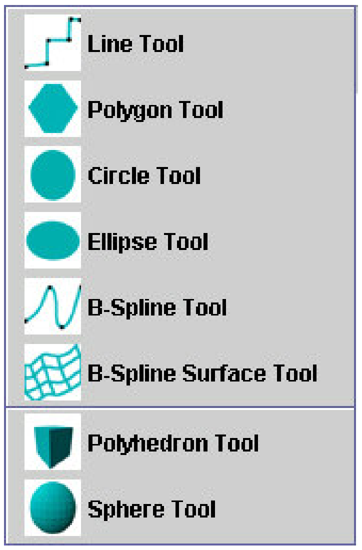


Figure 4: The Shapes menu which enables the creation of all the supported shapes

3.3. Tools

The tools button has 2 very essential tools. The first tool is the *transformer tool* which allows for translating, scaling, rotating, shearing and reflecting shapes. Providing all of these transformation operations in one window, will help to eliminate the cluttering of many windows in front of the 3D display. In addition, if a user selects several shapes for applying a transformation operation on, then these shapes don't have to be reselected if another transformation operation is required (e.g. a rotation followed by scaling). The transformation tool is discussed further in section 6. The second important tool is the *Shape Manager* which displays all of the shapes created. The *Shape Manager* provides two vital operations namely: deleting shapes, and showing the shape's data window. The Shape Manager is displayed in figure 5.

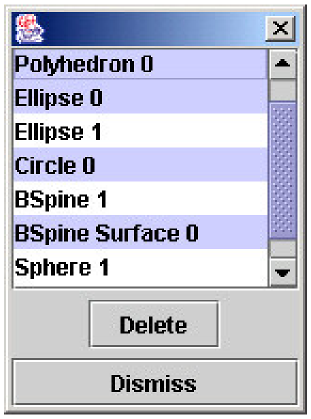


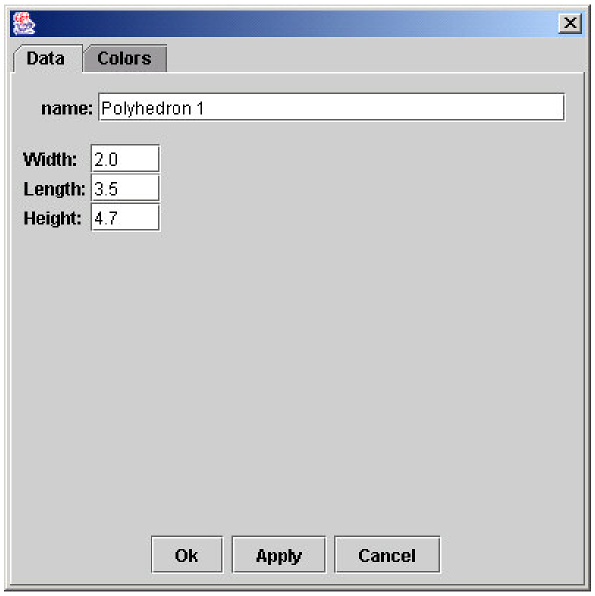
Figure 5: The Shape Manager

3.4. Toolbars

Two tool bars exist in the Zraphics main window, the toolbars provide easy access to both shape creation and the options available in the menu bar. A Zooming capability is also added to allow for easily zooming back and forth within the 3D display. The toolbars can also float for the convenience of users (i.e. can be pulled to any side of the window or become separated from the application).

3.5. Specifying the Shape Data

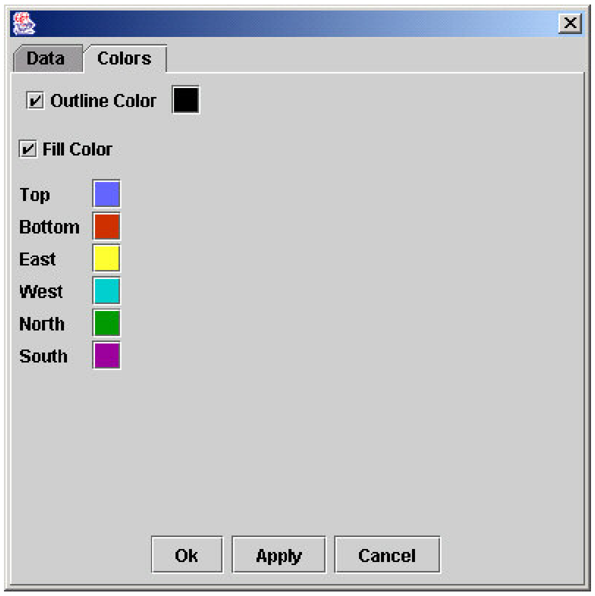
Upon clicking on a shape for creation, a window will pop up with 2 tabs. One for specifying the data required to create the shape such as the radius size of a sphere, or the control points for a curve. The second tab is for supplying the colors for both outlining and filling the shape. This window is consistent for all shapes, and is customized to provide the right parameters for each shape. In figure 6 the parameter window for a polyhedron for specifying the polyhedron's size and colors is displayed.



name: Polyhedron 1

Width: 2.0
Length: 3.5
Height: 4.7

(a)



☑ Outline Color ■

☑ Fill Color

Top	🟦
Bottom	🟥
East	🟨
West	🟦
North	🟩
South	🟪

(b)

Figure 6: The polyhedron parameter window, (a) the polyhedron data window, (b) the polyhedron color window.

4. Shapes

Zraphics supports the creation of several shapes. In the following sections, a detailed description of how these shapes are generated is discussed along with some illustrations.

4.1. Line

Lines are the most fundamental shape and are used as a basis for drawing more complicated shapes such as spheres and curves. The Bresenham algorithm for line drawing has been adopted to draw lines in Zraphics. Before discussing the Bresenham algorithm, let us focus on how to create a line in Zraphics. When clicking on the *Line Tool* icon in the toolbar (or selecting the *Line Tool* under the options menu button in the Zraphics menu bar), a window will appear prompting for the data and the colors to draw the line with. A line rendered in Zraphics is shown in figure 7.

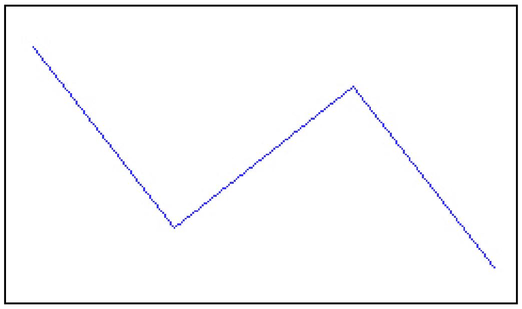


Figure 7: A line drawn in Zraphics

4.1.1. Bresenham's Line Drawing Algorithm

The beauty of this algorithm is that it only uses incremental integer calculations. This means that this algorithm is fast since there are no complicated computations involved. The details of deriving this algorithm however are beyond the scope of this report and can be found in [2]. In summary, the algorithm for drawing a line with a slope $|m| < 1$ is as follows:

1. Input the 2 line endpoints and store the left endpoint in (x_0, y_0).
2. Plot the point (x_0, y_0).
3. Calculate Δx, Δy, $2\Delta y$ and $2\Delta y - 2\Delta x$. Also calculate the starting value for the decision parameter $p_0 = 2\Delta y - \Delta x$.
4. At each x_k along the line, starting with k=0 do the following test
   - 4.1. If $p_k < 0$ then plot $(x_k + 1, y_k)$ and compute the next decision parameter as $p_{k+1} = p_k + 2\Delta y$
   - 4.2. Else plot $(x_k + 1, y_k + 1)$ and the new decision parameter is $p_{k+1} = p_k + 2\Delta y - 2\Delta x$
5. Repeat step 4 Δx times.

4.2. Polygon

Polygons can be drawn easily by selecting the *Polygon Tool*. The vertices for the polygon are required and should be supplied in the polygon data window. The polygon is rendered by drawing lines that connect its vertices. An example of a polygon which is filled and has a different colored outline is displayed in figure 8.

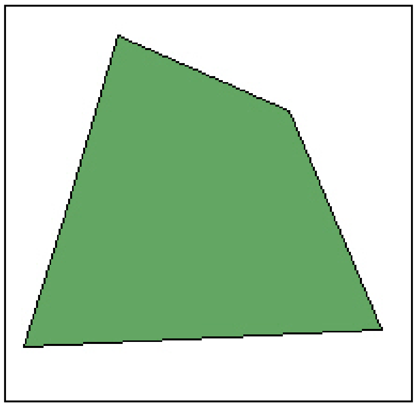


Figure 8: A filled polygon generated by Zraphics

4.3. Circle

Circles can be created by clicking on the *Circle Tool* and specifying circles radius in the circle data window. The world coordinates for the circle are computed using the following formulas:

$$x = r\cos\theta,\ 0 < \theta < 360$$

$$y = r\sin\theta$$

The number of points computed is proportional to the size of the radius. Since the circle is a 2D shape, the z component for the world coordinates will be fixed at zero and the same strategy will be followed for the rest of the 2D shapes. The generated world coordinates will be connected by lines. By connecting all the world coordinates together, means that a circle is some kind of polygon, the more vertices calculated leads to a smoother circle. However, calculating too many world coordinates will make Zraphics suffer due to the huge amount of expensive and unnecessary computation. Figure 9 shows a circle generated by Zraphics.

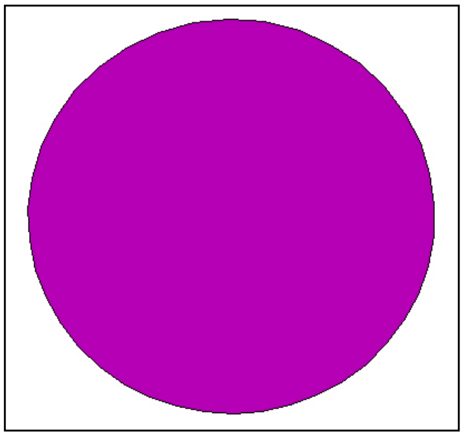


Figure 9: A filled circle generated by Zraphics

4.4. Ellipse

Similar to drawing a circle an ellipse is created by specifying the sizes of its radii r_x and r_y. The world coordinates are then computed by substituting in the following equations:

$$x = r_x \cos\theta \,,\; 0 < \theta < 360$$

$$y = r_y \sin\theta$$

An ellipse drawn in Zraphics is shown in figure 10.

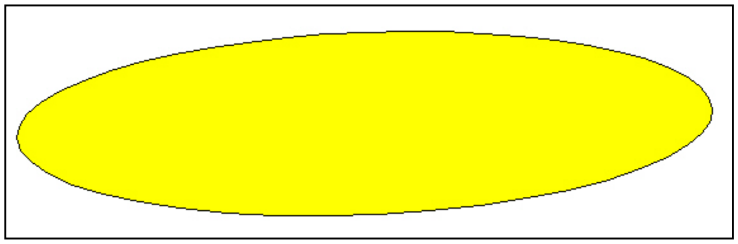


Figure 10: A filled ellipse generated by Zraphics.

4.5. Cubic B-Spline Curve

Uniform nonrational B-Spline curves are used to draw curves in Zraphics. A minimum of 4 control points are required to plot a single curve. For each 4 successive control points a fixed amount of subintervals are computed (23 subintervals in our case). These values can be generated by substituting in the following equation:

$$Q(t-t_i) = \frac{(1-t)^3}{6}P_{i-3} + \frac{3t^3-6t^2+4}{6}P_{i-2} + \frac{-3t^3-6t^2+3t+1}{6}P_{i-1} + \frac{t^3}{6}P_i$$

Where P_i denotes the ith control point and $0 < t < 1$. Q represents the curve segment and there is a total amount of curves equal to three less than the number of control points. After generating the points for each curve segment Q, they are joined by drawing lines between them. Figure 11 shows a B-Spline curve generated using 12 control points.

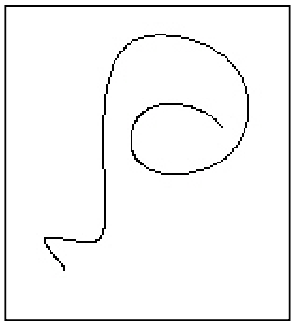


Figure 11: A B-Spline curve with 12 control points generated by Zraphics

4.6. Polyhedron

To create a polyhedron the width, length and height need to be supplied in the polyhedron data window. Six faces, which are basically polygons, are constructed from this information. The difference between filling a shape and filling a polyhedron is that a polyhedron accepts a different filling color for each face. A polyhedron is displayed in figure 12.

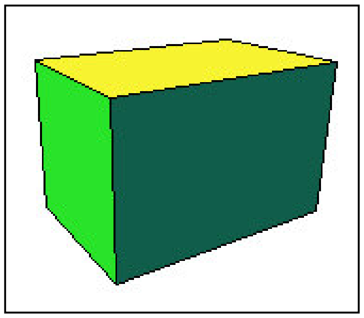


Figure 12: A filled polyhedron generated by Zraphics

4.7. Sphere

A sphere can be constructed and drawn in a similar fashion as to drawing a circle. All what's required by the user is to specify the radius of the sphere in the sphere data window. The world coordinates can be computed using the following equations:

$$x = r\cos\phi\cos\theta\,,\ -\Pi < \theta < \Pi$$

$$y = r\cos\phi\sin\theta\,,\ -\frac{\Pi}{2} < \phi < \frac{\Pi}{2}$$

Of course, the number of world coordinates is proportional to the radius size. A sphere is displayed in figure 13.

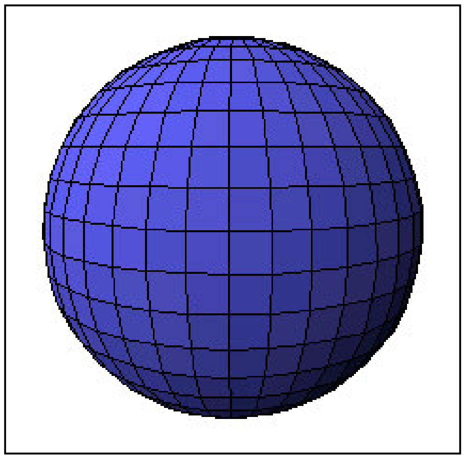


Figure 13: A filled sphere generated by Zraphics

4.8. Parametric Bicubic B-Spline Surface

B-Spline surfaces are constructed in a similar way to the construction of B-Spline curves. First, the horizontal control points are treated as control points for separate curves (i.e. each row for the surface is a single curve). After generating the curve segments for the rows, we deal with the newly generated world coordinates as new control points for separate curves in the vertical direction. Again, the curve segments in the vertical direction are computed. After obtaining the entire set of world coordinates, the final mesh is created by joining the world coordinates together. A B-Spline surface can be seen in figure 14.

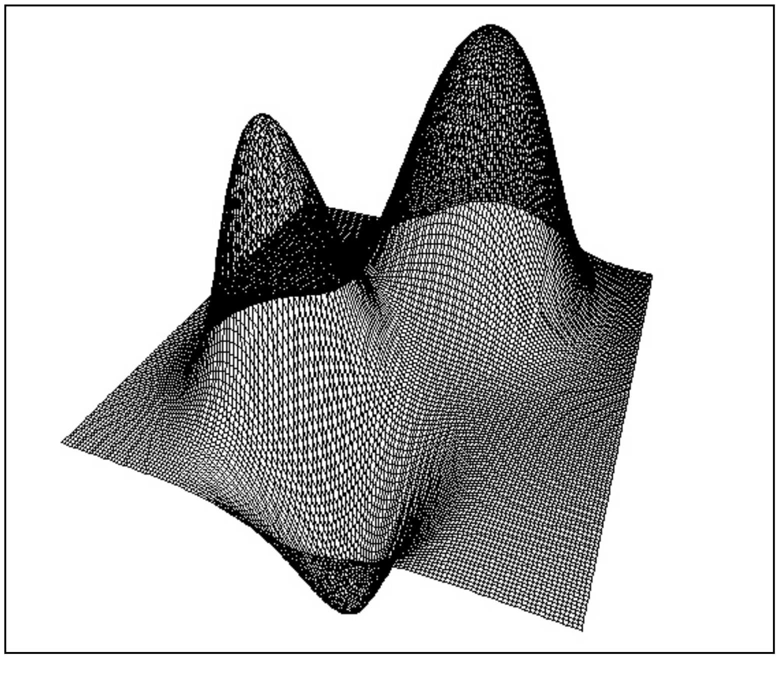


Figure 14: A B-Spline surface consisting of 81 control points generated by Zraphics.

5. Color Filling

Color filling shapes, which is supported by Zraphics, is the most important as well as the most difficult step. Why is it important? Because a representation of a 3D shape is not complete without color shading, in other words, to have a lighting effect on the shape to give it that live 3D look. This cannot be achieved without filling the shape first.Why is it difficult? Well let's discuss the design issues involved and let you be the judge of that.

5.1. Approach 1

Finding the pixels bounded by a specific shape (e.g. the interior of a circle) is required in order to successfully fill a shape. So the first step is to compute and store all the points bounded by a newly created shape. Whenever the shape is rotated or translated, the same transformation operation is applied to all the pixels stored. This approach worked perfectly in some cases and failed in others. In addition, this approach has 3 drastic disadvantages:

1. Lots of memory is required to store all the filling pixels for a shape, which can be very memory consuming.
2. It works fine if the size of the shape remains constant (rotated, reflected or translated). However, when the shape is scaled or sheared, then the filling pixels are not valid any more and have to be regenerated which can be difficult if many transformations have been previously applied on the shape.
3. It fails when trying to apply a hidden surface detection algorithm.

For these reasons, this method had to be abandoned (I guess this is the price paid for the sake of gaining experience!).

5.2. Approach 2

A simpler approach is to not worry about filling the shape until it is actually ready to be rendered on the screen. In other words, we do not deal directly with the real 3D word coordinates of the shape, but rather with the 2D screen coordinates calculated when projecting the shape on the screen (see section 7). When these screen coordinates are known, a polygon is created using these coordinates and filled. This approach not only eliminates the need to store and compute the filling pixels, but is also needed for the sake of detecting hidden surfaces (see section 8). Of course, this polygon is further divided into triangles, which is required for hidden surface detection, and each triangle is filled with the required color.

Approach 2 is used to fill the shapes created. All the shapes can be filled except for B-Spline surfaces which only works for some special cases (e.g. to have a 16 control point surface).

6. Transformations

No graphics package is complete without offering some basic transformation operations. The transformations available in Zraphics are: translation, rotation, scaling, shearing and reflection. By invoking the transformation tool the user can select the appropriate transformation and supply the required parameters. These transformations will be discussed in the sections to come.

6.1. Translation

Figure 15 shows the transformation tool with the translation tab selected. The user is prompted to specify the translation parameters for the x, y and z directions. Translation can be performed in Zraphics by multiplying the world coordinates of the shape to be translated with a 4x4 translation matrix. This translation matrix is of the form:

$$\begin{vmatrix} x' \\ y' \\ z' \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} x \\ y \\ z \\ 1 \end{vmatrix}$$

Where t_x, t_y and t_z represent the translation parameters for the x, y and z directions respectively. Figure 16 shows 2 wireframe polyhedrons; the red polyhedron represents the original (black) polyhedron after translating it 2 units in the x direction.

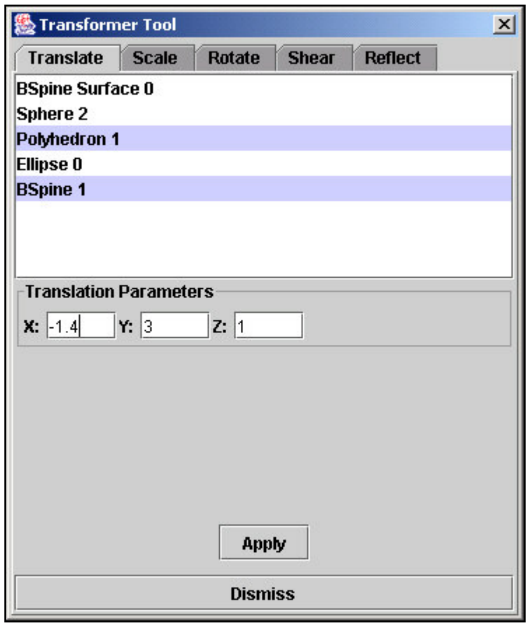


Figure 15: The transformer tool with the translation option selected.

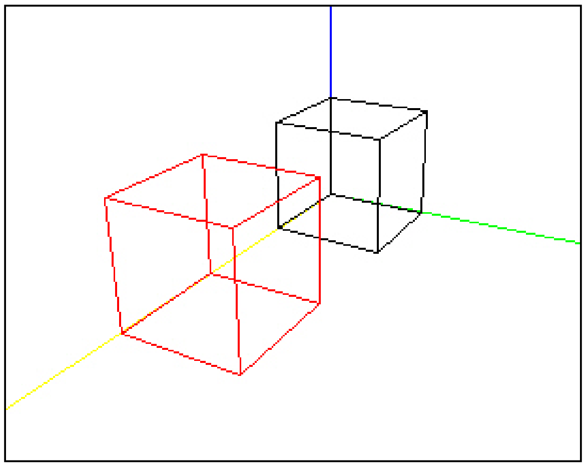


Figure 16: Translation of a polyhedron 2 units in the x-direction (the black polyhedron is the original, the red polyhedron is after the translation operation)

6.2. Rotation

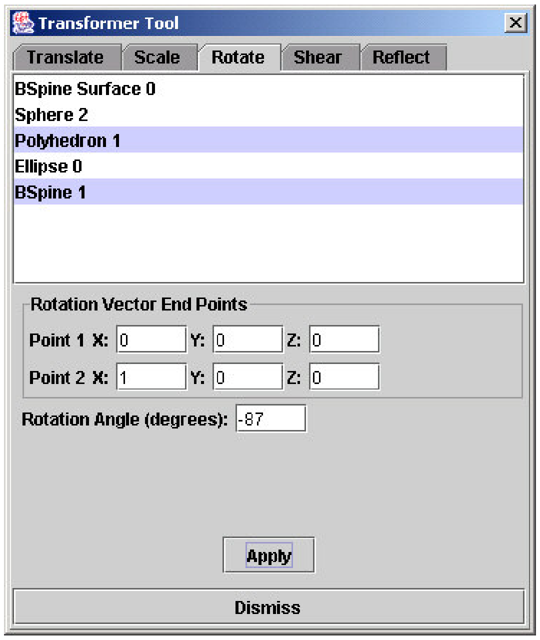
By selecting the rotation option in the transformation tool, the rotation axis and angle of rotation can be supplied to perform the required rotation (see figure 17). Rotating an object around an arbitrary axis in 3D is a bit more complicated than the previous simple translation operation. We will give a brief discussion on how to generate a rotation matrix around an arbitrary axis. For a detailed discussion, please see [1]. Rotation can be achieved by performing 5 steps:

1. Translate the shape to make the rotation axis pass through the origin.
2. Rotate the shape such that the rotation axis is aligned with one of the coordinate axis.
3. Rotate the shape the required rotation angle.
4. Apply the inverse rotation of step 2.
5. Apply the inverse translation of step 1.

Combining all of these steps in a single matrix is useful to speed up the computation needed to rotate a specific shape. The rotation matrix is as follows:

$$\begin{vmatrix} x' \\ y' \\ z' \\ 1 \end{vmatrix} = \begin{vmatrix} 1-2b^2-2c^2 & 2ab-2c\cos\frac{\theta}{2} & 2ac+2b\cos\frac{\theta}{2} & 0 \\ 2ab+2c\cos\frac{\theta}{2} & 1-2a^2-2c^2 & 2bc-2a\cos\frac{\theta}{2} & 0 \\ 2ac-2b\cos\frac{\theta}{2} & 2bc+2a\cos\frac{\theta}{2} & 1-2a^2-2b^2 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} x \\ y \\ z \\ 1 \end{vmatrix}$$

Where a, b and c are the components of the unit vector along the rotation axis. An example of a polyhedron about the x-axis with an angle of 135^0 is shown in figure 18 (the black polyhedron is before rotating and the red polyhedron is after performing the rotation operation).



BSpine Surface 0
Sphere 2
Polyhedron 1
Ellipse 0
BSpine 1

Rotation Vector End Points

	X	Y	Z
Point 1	0	0	0
Point 2	1	0	0

Rotation Angle (degrees): -87

Figure 17: The transformer tool with the rotation option selected.

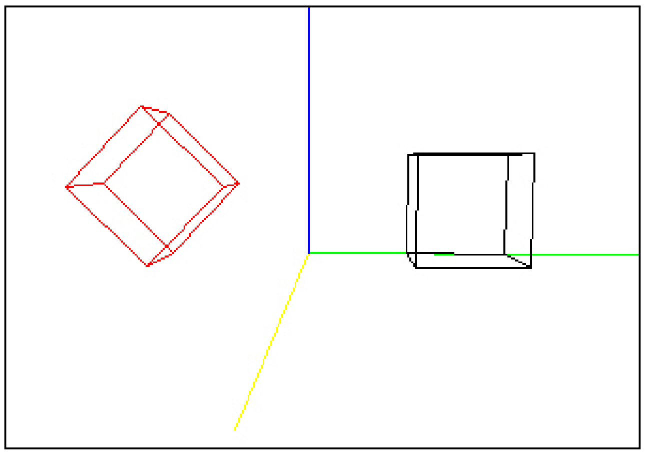


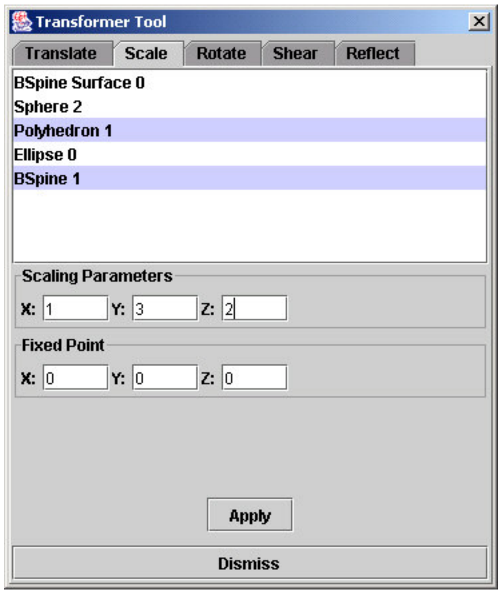
Figure 18: Rotation of a polyhedron about the x-axis with an angle of 135^0 (the black polyhedron is the original, the red polyhedron is after the rotation operation).

6.3. Scaling

Scaling a shape is an operation that is occasionally performed to change the shapes dimensions proportionally. This operation can be easily done in Zraphics by clicking on the scaling tab in the transformer tool as shown in figure 19. The user is expected to supply the scaling parameters in addition to an optional reference point. The scaling matrix is as follows:

$$\begin{vmatrix} x' \\ y' \\ z' \\ 1 \end{vmatrix} = \begin{vmatrix} s_x & 0 & 0 & (1-s_x)x_f \\ 0 & s_y & 0 & (1-s_y)y_f \\ 0 & 0 & s_z & (1-s_z)z_f \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} x \\ y \\ z \\ 1 \end{vmatrix}$$

Where s_x , s_y and s_z are the scaling parameters for the x, y and z directions. x_f , y_f and z_f are the coordinates of the fixed point. A Sphere scaled by a factor of 2 with respect to the y-coordinate is shown in figure 20.



BSpine Surface 0
Sphere 2
Polyhedron 1
Ellipse 0
BSpine 1

Scaling Parameters
X: 1 Y: 3 Z: 2

Fixed Point
X: 0 Y: 0 Z: 0

Figure 19: The transformer tool with the scaling option selected.

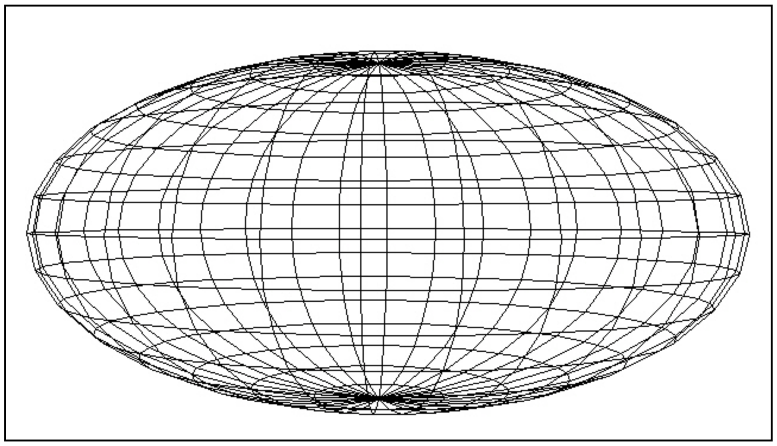


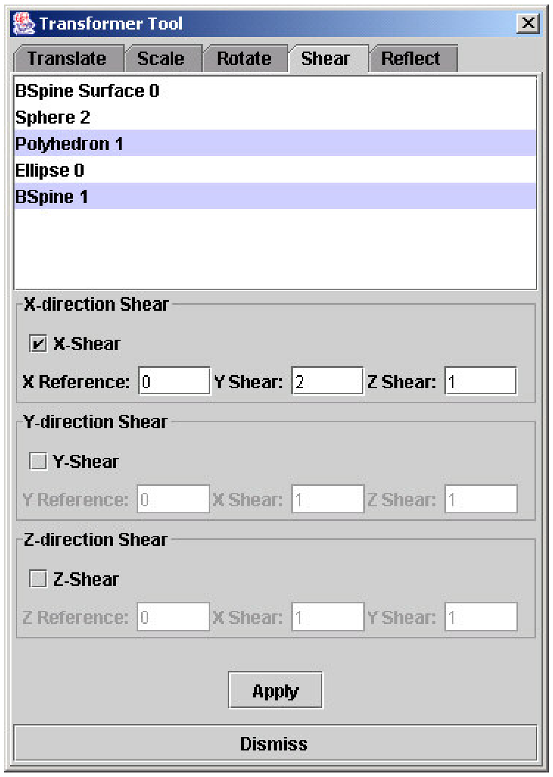
Figure 20: A Sphere scaled by a factor of 2 with respect to the y-coordinate.

6.4. Shear

The shear transformation allows for distorting a shape so that it appears as if the internal layers of the shape have slid over each other. This transformation can be applied in Zraphics by clicking on the shear tab in the transformer tool (as shown in figure 21). The shear option expects the shearing direction, reference point and shearing parameters. The shear matrix for a z-direction shear is as follows:

$$\begin{vmatrix} x' \\ y' \\ z' \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & sh_x & -sh_x . z_{ref} \\ 0 & 1 & sh_y & -sh_y . z_{ref} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} x \\ y \\ z \\ 1 \end{vmatrix}$$

Where sh_x and sh_y are the shear parameters for the x and y directions respectively, and z_{ref} is the z reference line. The shearing matrices for the x and y directions are similar. Figure 22 shows a sphere sheared in the x direction with shear value of 1 for both the y and z coordinates.


Transformer Tool

Translate | Scale | Rotate | Shear | Reflect

BSpine Surface 0
Sphere 2
Polyhedron 1
Ellipse 0
BSpine 1

X-direction Shear

☑ X-Shear

X Reference: 0 Y Shear: 2 Z Shear: 1

Y-direction Shear

☐ Y-Shear

Y Reference: 0 X Shear: 1 Z Shear: 1

Z-direction Shear

☐ Z-Shear

Z Reference: 0 X Shear: 1 Y Shear: 1

Apply

Dismiss

Figure 21: A : The transformer tool with the shear option selected.

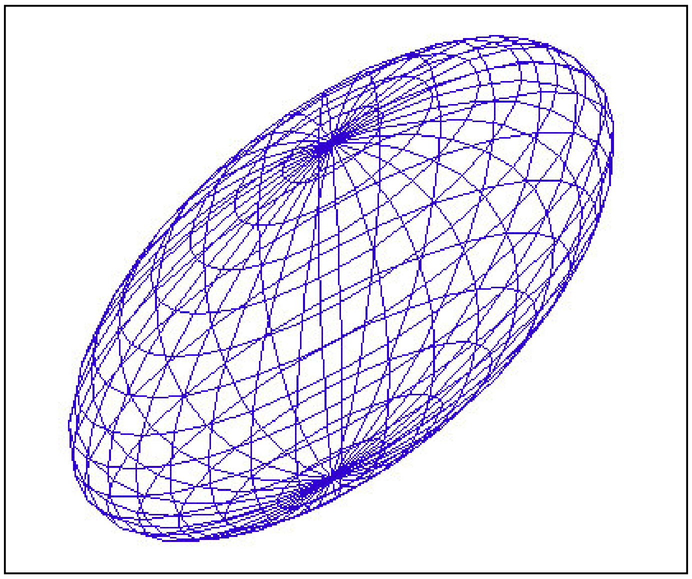
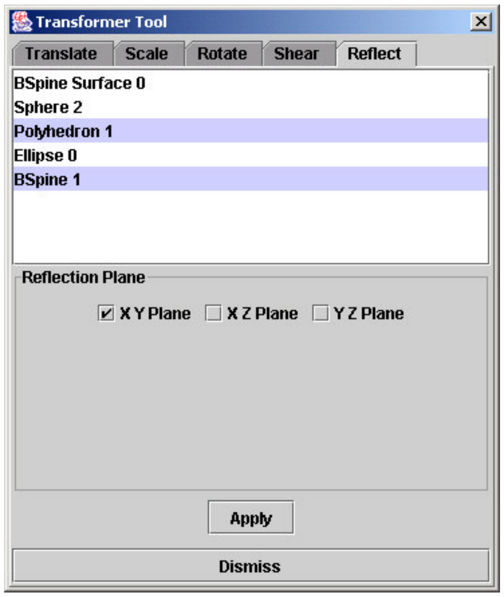


Figure 22: A sphere sheared in the x direction with shear value of 1 for both the y and z coordinates.

6.5. Reflection

Reflection is a nice option, which enables a shape to be reflected relative to a plane. The reflection operation can be performed by selecting the reflection tab in the transformer tool and choosing the plane to reflect on (see figure 23). The reflection matrix for a reflection relative to the XY plane is as follows:

$$\begin{vmatrix} x' \\ y' \\ z' \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} x \\ y \\ z \\ 1 \end{vmatrix}$$



BSpine Surface 0
Sphere 2
Polyhedron 1
Ellipse 0
BSpine 1

Reflection Plane

[✓ X Y Plane] X Z Plane Y Z Plane

Figure 23: A : The transformer tool with the reflection option selected.

7. Viewing Projections

Two types of viewing projections are supported by Zraphics. These viewing projections are the orthographic parallel projection and perspective projection. Before discussing the details for obtaining these types of viewing projections, let us define a mapping between the world coordinates for the shape and the screen coordinates. This mapping can be accomplished in 2 stages. The first stage is the viewing transformation that transforms the world coordinates to get the eye coordinates. The eye coordinates are basically the transformation of the world coordinates onto a screen that is located between the shape and the eye. The second stage is to transform the 3D eye coordinates onto the final 2D screen coordinates. Below is a brief discussion on how to generate the viewing projections, for more information please see [3].

7.1. Viewing Transformation

The viewing transformation is used to perform both types of viewing projections. We imagine that a screen exists between the eye E and the shape. We now have 2 coordinate systems namely: the world coordinate system (with O as its origin) and the eye coordinate system (with E as its origin). Before listing the three steps needed to perform the viewing transformation, let us define the viewpoint E by its spherical coordinates, such that:

$$x_e = \rho \sin\phi \cos\theta$$

$$y_e = \rho \sin\phi \sin\theta$$

$$z_e = \rho \cos\phi$$

The viewing transformation is accomplished with the following 3 steps:

1. Move the origin from O to E.
2. Rotate the coordinate system about the z-axis.
3. Rotate the Rotate the coordinate system about the x-axis.

The result of these steps is the following:

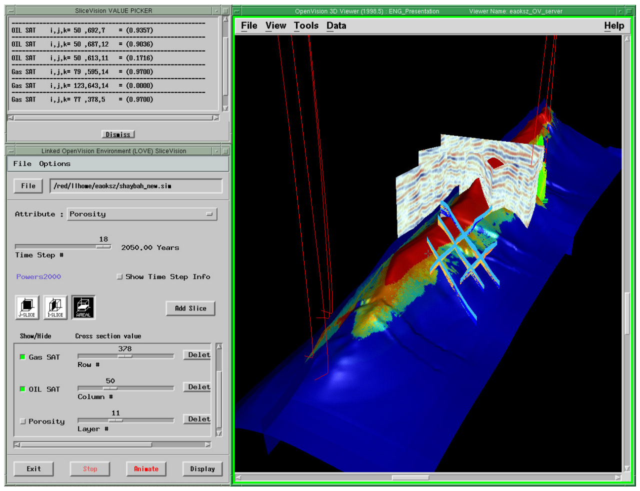
$$\begin{vmatrix} x_e & y_e & z_e & 1 \end{vmatrix} = \begin{vmatrix} -\sin\theta & -\cos\phi\cos\theta & \sin\phi\cos\theta & 0 \\ \cos\theta & -\cos\phi\sin\theta & \sin\phi\sin\theta & 0 \\ 0 & \sin\theta & \cos\phi & 0 \\ 0 & 0 & -\rho & 1 \end{vmatrix} \begin{vmatrix} x_w & y_w & z_w & 1 \end{vmatrix}$$

Where the first matrix is the eye coordinates, the second matrix is the viewing transformation matrix, and the third matrix is the world coordinate matrix.

7.2. Orthographic Parallel Projection

Orthographic parallel projections are often used in many scientific and engineering 3D applications. An example of these applications is *OpenVision* and *StrataModel*, which are 2 well known engineering applications used in the petroleum industry, for visualizing 3D geological models (see figure 24). The beauty of this type of projection is that parallel lines remain parallel in the display, which means that the dimensions of objects displayed remain proportional.

After computing the eye coordinates using the viewing transformation discussed earlier; the 2D screen coordinates can be easily generated by simply ignoring the z_e component. That is, $x = x_e$ and $y = y_e$. A polyhedron projected using this type of projection is shown in figure 25.


```
OIL SAT    i,j,k= 50 ,692,7    = (0.9357)
OIL SAT    i,j,k= 50 ,687,12   = (0.9036)
OIL SAT    i,j,k= 50 ,613,11   = (0.1716)
Gas SAT    i,j,k= 79 ,595,14   = (0.9700)
Gas SAT    i,j,k= 123,643,14   = (0.0000)
Gas SAT    i,j,k= 77 ,378,5    = (0.9700)
```
File: /red/llhome/eaoksz/shaybah_new.sim

Attribute : Porosity

Time Step # 18 — 2050.00 Years

Gas SAT — Row # 378

OIL SAT — Column # 50

Porosity — Layer # 11

Figure 24: A screen shot of OpenVision that is a 3D visualization application used in the petroleum industry to visualize seismic volumes and geological surfaces.

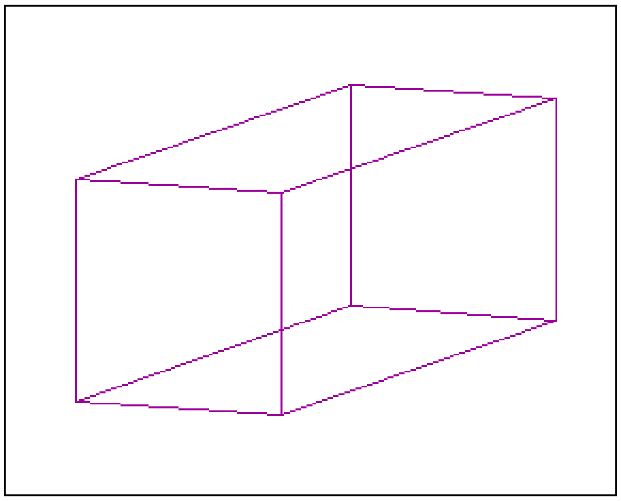


Figure 25: A polyhedron projected using orthographic parallel projection.

7.3. Perspective Projection

A perspective projection is performed if the shape to be projected is to have a complete 3D effect where a perception of depth exists. Here parallel lines are not necessarily parallel when projected onto the 2D screen, but rather meet at a vanishing point. All of the vanishing points lie on the same line, which is called the horizon.

By using the viewing transformation, the 2D screen coordinates can be computed as: $x = -d\frac{x_e}{z_e}$ and $y = -d\frac{y_e}{z_e}$. Where d is the minimum of the length and width of the 2D screen. A polyhedron projected using the perspective projection is shown in figure 26.

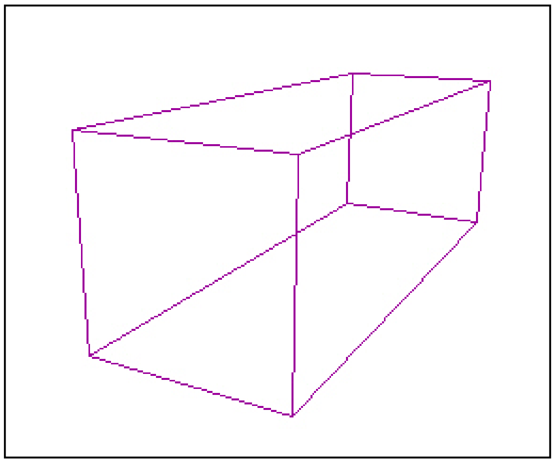


Figure 26: A polyhedron projected using perspective projection.

8. Hidden Surface Detection and Hidden Line Elimination

This section deals with two issues. The first issue is hidden surface detection, this is needed to evaluate the surface of a shape and to determine if it is invisible, partially obscured or totally visible when projected onto the 2D screen display. The second issue is to eliminate any hidden lines, which is similar to hidden surface detection, but deals with lines instead of surfaces. Both these techniques are supported in Zraphics and are going to be discussed next.

8.1. Hidden Surface Detection

Hidden surface detection (also known as hidden face elimination) is a technique used to detect which parts of a surface are hidden. This technique is also interested in detecting the portion of a visible surface, if it was partially covered by another shape. Zraphics supports hidden surface detection by implementing a well known algorithm called the painter's algorithm which is explained briefly in next section. More details of the painter's algorithm are mentioned in [3].

8.1.1. Painter's Algorithm

The painter's algorithm takes advantage of the triangles created for filling a shape (see section 5.2). Each triangle consists of 3 vertices and these vertices are stored in anticlockwise order. The order of these vertices is very important and is used to determine the normal vector of this triangle (whether it is pointing with or against the viewpoint). Depending on the orientation of the triangle's vector, 2 decisions can be made:

1. If the normal of the triangle is pointing in the same direction as the viewpoint means that it is a back face and therefore should not be rendered.
2. If the normal is in the opposite direction of the viewpoint means that this triangle is visible and has to be considered.

Once a decision has been made about a specific triangle (if it is hidden or visible), further processing needs to be done for the visible triangles. This processing is interested in finding the visible portions of a triangle if more than one shape is to be drawn. The painter's algorithm solves this problem by following the same technique followed by painters (hence its name) where further objects are drawn first then the ones nearer and so on. This can be computed cheaply by sorting all the triangles according to their depth. The depth of a triangle can be computed by taking the average depth of its vertices. After obtaining the depth of all the triangles, they are sorted then rendered to the screen by beginning to draw the farthest triangle first. Since the farthest triangle is going to be drawn to the screen first, means that it will not obscure any triangle in front of it. On the other hand, any triangle behind it may be partially or totally obscured. The beauty of this method is its efficiency, speed and relatively simple implementation. However, the painter's algorithm may fail in some special cases. An example of the output of this algorithm is shown in figure 27, where a sphere is partially obscuring a polyhedron.

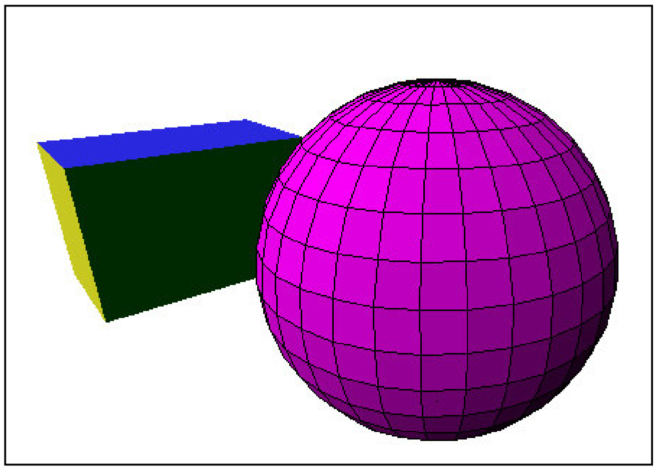


Figure 27: A sphere is partially obscuring a polyhedron using the painter's algorithm.

8.2. Hidden Line Elimination

This method is interested in finding if a line is completely visible, invisible or partially visible. It is different from the painter's algorithm since it deals with line segments instead of triangles. To achieve this task we need to perform several tests on each line, comparing it with all the triangles to render, to determine its visibility. We will use P and Q to denote the endpoints of a line, and will use A, B and to denote the vertices of a triangle. The tests are summarized in the following 8 points (for further information please refer to [3]).

1. Check if the line segment PQ is entirely on one side of the triangle ABC. If it is, then draw the line, otherwise proceed to test 2.
2. Check if P or Q is not further than any of the vertices ABC. If it is, then draw the line, otherwise proceed to test 3.
3. Check if P and Q lie on one side of the line AB and C lies on the other side. This means that the line PQ is not obscured by the triangle ABC (the other lines of the triangle have to be computed as well). If it is, then draw the line, otherwise proceed to test 4.
4. The triangle ABC doesn't obscure the line PQ if the vertices ABC lie on the same side of the infinite line through P and Q. If this is the case, then draw the line, otherwise proceed to test 5.
5. Test if neither P nor Q lies behind the infinite plane through A, B and C. If this is the case, then draw the line, otherwise proceed to test 6.
6. Test if both P and Q are behind the triangle ABC. If this is the case, then don't draw the line, otherwise proceed to test 7.
7. Test if P is nearer than ABC and if P is inside the triangle ABC, then PQ is visible. The same test should be done for Q. If the is the case, then draw the line, otherwise proceed to test 8.
8. Compute the intersections I and J of PQ with ABC in 2D. If, in 3D, such an intersection lies in front of ABC, this triangle does not obscure PQ. Otherwise, the intersections lie behind and this triangle obscures part of PQ, then create a new line segment I and J and go back to the first test.

Figure 28 (a) shows a curve that is not obscured by a polyhedron, figure 28 (b) shows the same scene but rotated so that the curve is partially obscured by the polyhedron.

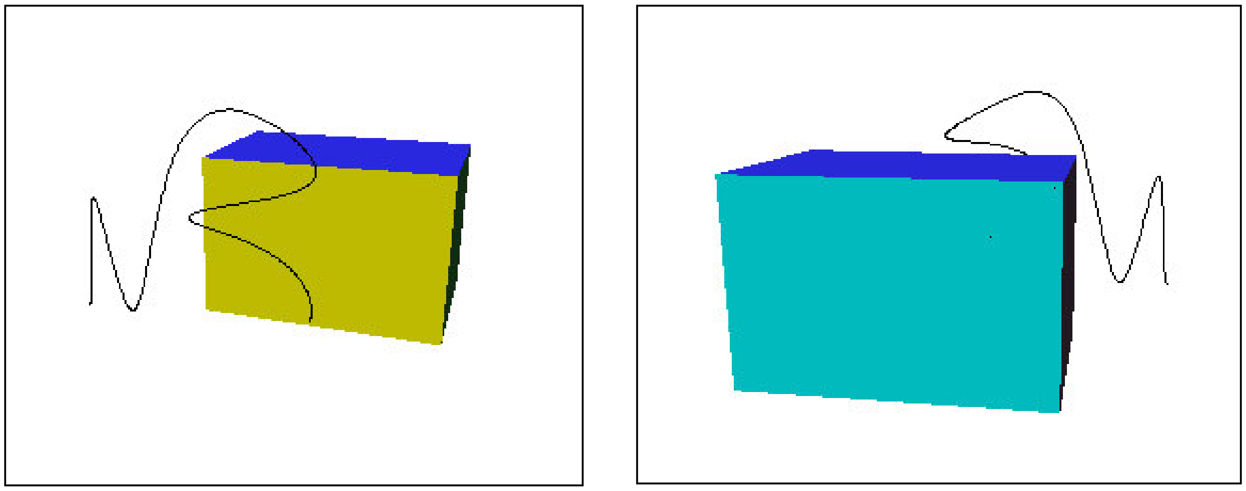

(a) (b)

Figure 28: Hidden line detection, (a) a curve displayed in front of a polyhedron, (b) same as (a) but the scene is rotated so that the polyhedron is partially obscuring the curve.

9. Colour Shading

Colour shading is used to give a lighting (shading) effect to surfaces. The surface is brighter if it is pointing towards the source of light and is darker otherwise. Colour shading is implemented in Zraphics. To perform colour shading, 2 vectors are required. The first vector is a unit vector directed from the source of light towards the triangle (the triangles generated for filling colours). The second vector is the normal vector of the triangle that is perpendicular to the triangle and pointing more or less towards the viewpoint. By computing the inner product for the 2 vectors and since the range of all inner products are known, a percentage can be assigned to the colour for each triangle.

10. Future Work

Zraphics provides various functionalities for the creation and manipulation of 2D and 3D shapes. It has been designed in a way to make it straight forward for future enhancements (as stated in the design section). The functionality in addition to the performance can be enhanced in several areas and are stated in the following points:

1. Java supports the capability of multithreading. This capability can be taken advantage of by splitting the display into several buffers. Each thread can now be assigned the task of computing the required data to be projected in the screen. This will be extremely beneficial if multiple CPU's are available. It will also help in the case of using a single CPU since the entire screen does not need to be rendered in some cases.
2. Java Native Interface is a java capability, which allows java to call native code such as C, C++ and FORTRAN. This is helpful if a large amount of computation is required where it is better done using a lower level, and much faster language like "C" and "Assembly".
3. Zraphics supports one family of curves, which is the Uniform Nonrational B-Spline curves. Other curves can be adopted like the Nonuniform Rational B-Splines, Bezier curves and Hermite curves.
4. The painter's algorithm, which is used for hidden surface detection, is an efficient and fast method. However, this method might fail in some special cases. Other methods can be used such as the Z-Buffer algorithm.
5. Currently, Zraphics allows for creating shapes by clicking on the required shape tool and prompts the user to provide the data manually. This mechanism is some times tedious. A better way is to have the selection of world coordinates to be interactive, where the user simply selects the data points on the screen using the mouse.

11. Conclusion

In the this report, the shapes supported by Zraphics were discussed along with the mechanism used to generate them. These shapes are: Lines, polygons, circles, ellipses, B-Spline curves, B-Spline surfaces, polyhedrons and spheres. The shapes can be displayed as wireframes or filled with a specific colour. Several transformation operations were also mentioned which are: translation, scaling, rotation, shearing and reflection. Two viewing projections have also been mentioned which give a 3D effect when displaying the shapes. The viewing projections are the orthographic parallel projection and the perspective projection. We also took a look at Zraphics's the capability to detect both hidden surfaces and hidden lines. The issue of colour shading was also touched upon. The report also listed some items that may be considered for future work.

Acknowledgements

I would like to thank and express my gratitude to Dr. Muhammad Sarfraz for his valuable suggestions, guidance and help that improved the overall quality of this project.

References

[1] J. Foley, J. Hughes, A. Van Dam, "Computer Graphics", Addison Wesley Professional, 1995.

[2] D. Hearn, M. Baker, "Computer Graphics C Version", Printice Hall, 1997.

[3] L. Ammeraal, "Computer Graphics for Java Programmers", John Wiley & Sons Ltd., 1998.