Q.1

c. Write the open GL commands to plot a sphere of radius 3 units and rotate it around the diameter along line x = y in x-y plane.

Answer:

void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);
void glutWireSphere(GLdouble radius, GLint slices, GLint stacks);

radius: The radius of the sphere.
Slices: The number of subdivisions around the Z axis (similar to lines of longitude).
stacks: The number of subdivisions along the Z axis (similar to lines of latitude).

Description: Renders a sphere centered at the modeling coordinates origin of the specified radius. The sphere is subdivided around the Z axis into slices and along the Z axis into stacks.

d. Explain the concept related to C^2 continuity.

Answer:

Smoothness can be described by degree of continuity

- zero-order (C^0): position matches from both sides
- first-order (C^1): tangent matches from both sides
- second-order (C^2): curvature matches from both sides
- *Gn*vs. *Cn*zero order first order second order

Parametric continuity (*C*) of spline is continuity of coordinate functions Geometric continuity (*G*) is continuity of the curve itself.

e. Find the transformation matrix for rotating (x, y, z) along X-axis by an angle 30^0.

Answer:

This 3D rotational transformation matrix for a point around X-axis.
Replace angle with 30^0.

f. What is vanishing point? Which type of projections uses vanishing point?

Answer:

Parallel lines in the world intersect on the paper at a point known as the vanishing point. Each set of lines has a different vanishing point. But, just what is a vanishing point? Projective geometry sheds light on this issue.

Because image formation is the projection from a 3D world to a 2D surface, each point on the image plane is the projection of an infinite number of points in the world. Usually, the closest point is opaque and therefore we think of the point on the image plane as being the projection of only one point in the world. However, ideal points in the world (i.e., ideal points in $\mathcal{P}^3$), always project onto the image plane regardless of the opacity of other points. Each point on the image plane is the projection of an ideal point. To see this, consider the following perspective

projection equation (a general projective transformation is used for simplicity) fro
an ideal point $[X,Y,Z,0]^T$ in the world to a point $[x,y,w]^T$ in the image plane:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 0 \end{bmatrix}.$$

Because the matrix is full rank, each ideal point projects to a different point on the image plane. Since parallel lines in 3D space intersect at an ideal point in $\mathcal{P}^3$, their projections in the image plane must still intersect at a point. But now, through the projection matrix, the ideal point has become a ``real" point, in the sense that it is no longer ideal. However, some ideal points do not become ``real" points. If the ideal point represents a direction that is parallel to the image plane, then the dot product of the ideal point with the third row of the matrix (which is the *z* axis of the plane) is zero, and the projected point is still ideal. Just as the ideal points of $\mathcal{P}^2$ have a one-to-one correspondence with all the points in $\mathcal{P}^1$, so the ideal points of $\mathcal{P}^3$ have a one-to-one correspondence with all the points in $\mathcal{P}^2$. Thus, we see that the fact that parallel lines in the world intersect when drawn on a piece of paper follows naturally from projective geometry.

g. Why hidden surfaces are removed while rendering a solid on the output screen? Write the name of two algorithms that are used to achieve this.

Answer:

Hidden surface removal (HSR) determines which polygons are nearest to the viewer at a given pixel Key criterion: a point *P occludes* a point *Q* (and thus *Q* is "hidden") if *P* and *Q* lie on the same ray (line) from the camera or eye and *P* is between the camera location and *Q*

Calculating this ray is tough with a frustum, but normalizing that frustum to a cube (which the projection matrix does) transforms the oblique rays to straightforward parallelism with the *z* axis.

Two algorithms may be: Painter's algorithm and Depth Buffer Algorithm

Q.2 b. Write Scan line filling algorithm and use the algorithm to fill the inside area of the polygon bounded by (1, 1), (4, 4) and (7, 1).

Answer:

The basic algorithm is:

Given a polygon *P*, with edges $e_1, e_2, \ldots, e_n$

1. Use DDA or Bresenham's algorithm to find each pixel on *P*'s boundary
2. Sort the pixels in scan-line order, storing them in a list *L*
3. Pull off pairs of points $(x_1, y_1), (x_2, y_2)$ from *L* and color each pixel (x, y) such that

$$x_1 \leq x \leq x_2$$

(Note it must be the case that $y_1=y=y_2$)

You've been lied to, scan-lines that hit polygon vertices intersect the region an odd number of times and horizontal edges have infinitely many intersections with a scan-line Corrections are:

- Don't process horizontal edges
- When a scan-line hits a vertex
  - record two intersection when the vertex is a local maxima or local minima
  - record one intersection when the vertex is between edges that continue increasing or decreasing

For a high resolution device there may be thousands of intersections, so even an $n \log n$ sort routine may take time Bucket sorting (a form of hashing) can save time.

Q.3 a. What do you mean by homogeneous coordinate system? How does it help in finding a composite matrix for different transformations applied in a sequence? Explain your answer with a suitable example.

Answer:

One of the many purposes of using homogeneous coordinates is to capture the concept of infinity. In the Euclidean coordinate system, infinity is something that does not exist. Let us consider two real numbers, *a* and *w*, and compute the value of *a/w*. Let us hold the value of *a* fixed and vary the value of *w*. As *w* getting smaller, the value of *a/w* is getting larger. If *w* approaches zero, *a/w* approaches to infinity! Thus, to capture the concept of infinity, we use two numbers *a* and *w* to represent a value *v*, *v=a/w*. If *w* is not zero, the value is exactly *a/w*. Otherwise, we identify the infinite value with (*a*, 0). Therefore, the concept of infinity can be represented with a number pair like (*a*, *w*) or as a quotient *a/w*.

For example, suppose we have a line $Ax + By + w = 0$. Dividing the equation by w and assuming *x* and *y* having w a factor, it yields $A(x/w) + B(y/w) + 1 = 0$. or

$$Ax' + By' + 1 = 0.$$

The coordinate (x, y) is then represented as (x', y', 1) or in more general (x, y, 1). This works for three-dimension as well. One can replace a point (*x, y, z*) with (*x, y, z, 1*). This is of great help in expressing transformation as product of matrices.

An example of combination of basic transformations to get a combined transformational effect will complete the answer.

b. Using mid-point subdivision method, clip a line segment between lines (1, 1) and (8, 12) so that it can be displayed within a rectangular window bounded by (2, 1) and (12, 10).

Answer:

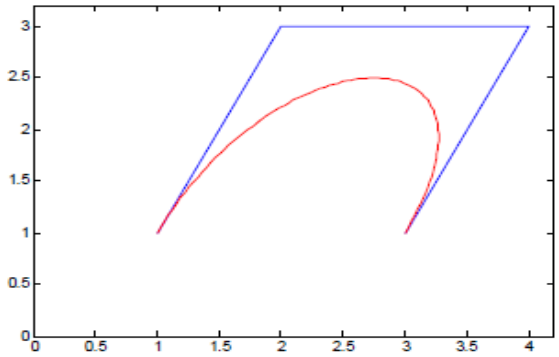
The clipped line segment is between (2, 18/7) and (74/11, 10)

Q.4 b. Determine the equation of Bezier curve over the interval for t=0: .01: and control points are at (1, 1), (2, 1), (4, 3) and (3, 1).

Answer:

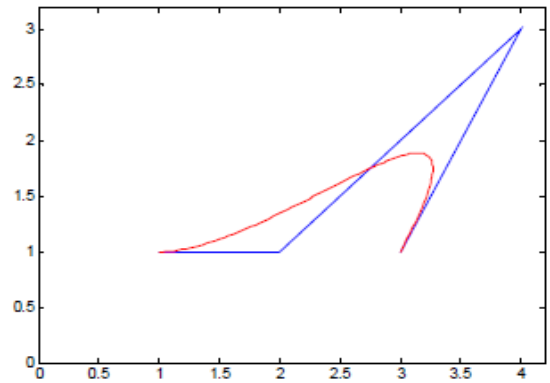
One snap shot is as below:

```
% Bezier curve for n=3. Example 5.7 R&A
t=0:.01:1;
x=[1 2 4 3];
y=[1 3 3 1];
px=(1-t).^3*x(1)+3*t.*(1-t).^2*x(2)+3*t.^2.*(1-t)*x(3)+t.^3*x(4);
py=(1-t).^3*y(1)+3*t.*(1-t).^2*y(2)+3*t.^2.*(1-t)*y(3)+t.^3*y(4);
plot(x,y); hold
plot(px,py,'r');
axis([0 4.2 0 3.2]);
```



X(t)	1.0	1.326	1.688	2.062	2.424	2.75	3.016	3.198	3.272	3.214	3.0
Y(t)	1.0	1.54	1.96	2.26	2.44	2.5	2.44	2.26	1.96	1.54	1.0

```
% Bezier curve for n=3. Example 5.7 R&A, one data pt altered
t=0:.01:1;
x=[1 2 4 3];
y=[1 1 3 1];
px=(1-t).^3*x(1)+3*t.*(1-t).^2*x(2)+3*t.^2.*(1-t)*x(3)+t.^3*x(4);
py=(1-t).^3*y(1)+3*t.*(1-t).^2*y(2)+3*t.^2.*(1-t)*y(3)+t.^3*y(4);
plot(x,y); hold
plot(px,py,'r');
axis([0 4.2 0 3.2]);
```



1	1.326	1.688	2.062	2.424	2.75	3.016	3.198	3.272	3.214	3
1	1.054	1.192	1.378	1.576	1.75	1.864	1.882	1.768	1.486	1

Q.5 a. What are the issues involved in 3D clipping? How is it different from 2D clipping? Describe any one algorithm to clip 3D object.

Answer:

Clipping is very important in 3D graphics. The main purpose is to prevent the triangle drawing routine from trashing memory and drawing out of the screenspace. 3D-clipping can also help you to speed up your rendering engine. I'll first explain the basics and at the end you'll find some ideas for the speed issue.

Basically there are two different approaches to clipping. The first and most common is the 2D clipping algorithm. In this kind of algorithm the clipping is done at the last stage of rendering. The triagle-routine used to render the polygons onto the screen make sure that you don't draw outside the screen. This can be very fast, but increases the complexity of the triangle-filler (and it's not as easy as 3D-clipping can be). This clipping method works directly with two-dimensional screen-coordinates.

In 3D clipping you do everything in 3D space. (In my examples it's done in camera-space. If you don't know about this don't worry about it.) If we want to clip in 3D space we need to have a description of the stuff the camera of our rendering-engine can see. When we use perspective projection (the most common projection type) this space can be described as a frustum.

Extend Cohen-Sutherland and Sutherland Hodgman algorithm for this purpose.

b. When a projection is called a cabinet projection and a cavalier projection? Determine the projection matrix for cabinet and cavalier projections.

Answer:

cos(phi) = Xp – x / L
sin(phi) = Yp – y / L
Xp = x + L cos(phi)
Yp = y + L sin(phi)

Length L depends on the angle alpha and the z coordinate of the point to be projected:
tan (alpha) = z / L
The transformation matrix for producing any parallel projection onto the xy plane can be written as

$$\text{Parallel } M = \begin{bmatrix} 1 & 0 & L1\cos(phi) & 0 \\ 0 & 1 & L1\sin(phi) & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Two special cases of oblique projection
1) Cavalier
2) Cabinet

1) Cavalier

Alpha = 45°, tan (Alpha) = 1 => L1 = 1 this is a **Cavalier** projection such that all lines perpendicular to the projection plane are projected with no change in length.

2) Cabinet

tan (Alpha) = 2, Alpha= 63.40°, L1 = 1 / 2

Lines which are perpendicular to the projection plane are projected at 1 / 2 length. This is a **Cabinet** projection

Q.6 a. Write the Painter's algorithm for back face detection.

Answer:

The painter's algorithm, sometimes called depth-sorting, gets its name from the process which an artist renders a scene using oil paints. First, the artist will paint the background colors of the sky and ground. Next, the most distant objects are painted, then the nearer objects, and so forth. Note that oil paints are basically opaque, thus each sequential layer completely obscures the layer that its covers. A very similar technique can be used for rendering objects in a three-dimensional scene. First, the list of surfaces are sorted according to their distance from the viewpoint. The objects are then painted from back-to-front.

While this algorithm seems simple there are many subtleties. The first issue is which depth-value do you sort by? In general a primitive is not entirely at a single depth. Therefore, we must choose some point on the primitive to sort by.

Implementation:

The algorithm can be implemented very easily. First we extend the drawable interface so that any object that might be drawn is capable of supplying a z value for sorting.

```
import Raster;
        public abstract interface Drawable {
          public abstract void Draw(Raster r);
          public abstract float zCentroid();
        }
```

Next, we add the required method to our triangle routine:

```
        public final float zCentroid() {
          return (1f/3f) * (vlist[v[0]].z + vlist[v[1]].z + vlist[v[2]].z);
        }
```

b. Describe the Ray Tracing method of surface rendering.

Answer:

Ray tracing is a versatile technique that uses the same model to integrate aspects of light/object interaction that were previously handled by separate ad hoc algorithms - reflections, hidden surface removal and shadows.

The idea of ray tracing is tracing the light for each pixel, from an eye or view point through the pixel and into the scene.

Description

The method to trace a light from light source and propagate to eye or view point, is called 'backwards ray tracing'. There are infinity of rays emanating from light source and it is difficult, therefore, to trace rays in the direction of light propagation.

In the implementation we trace the rays backward from the view point through each pixel and into these scene. That is we trace in the reverse direction of light propagation. This is because we are only interestd eventually in a fixed number of rays - those that pass through the view plane - say, one per pixel.

Ray/objects intersection and color rendering are two major issues need to be evaluate carefully. There are many existing algorithm to compute the intersection of light with objects. To select a sufficient method for ray/object intersection has a significant effect of performance, because most of time for raytracing is sepnd for intersection. Due to different objects have different characteristics, it is not necessary to apply the same ray/object intersection method for different objects. For example, even the geometric method may good enough for ray/sphere intersection evaluation, but it may not sufficient for ray/quardrics intersection.

After find out the intersec point for ray and objects, the next issue is render the color for that point. Following formula is used to model the color by ray tracing.

```
I = Ilocal + Krg . Ireflection + Ktg . Itransmitted
```

`Where` I is the final color of rendering
Ilocal is the term by direct illumination
Krg is the coefficient of reflection of object
Ireflection is reflected light
Ktg is the cofficient of transmitted or refraction of object
Ireflection is trasnmitted or refracted light

Phong model is a good model for local light rendering. Reflection and transmitted light need apply some geometric methods and need find out the normal vector of light vector at intersection point.

Q.7 Write a short note on any <u>TWO</u> of the followings:

(i) Self similar fractals.
(ii) Specular Reflection
(iii) Simulating acceleration in animation

Answer:

(a)

On the left, we see a Julia set. I have drawn red brackets to show where part of the set is endlessly repeated, smaller and smaller and smaller. This self-similarity is usually the major identifying feature of a fractal. Mandelbrot originally defined fractals using the "fractal dimension," which I will write about some day. A much simpler definition involves a kind of loose self-similarity.

An example of this self-similarity is a cloud. Without other objects to compare with, it is very hard to tell the size of a cloud by looking at a photo. Is it a small picture of a big cloud or a big picture of a small cloud, or what? Different parts of the cloud, of vastly different sizes, are more or less similar to each other.

The self-similarity of this fractal fern is obvious. Some of the parts are mirror images of other parts, as well as being bigger or smaller.
The self-similarities of the above fractals are exact. Each smaller portion is an exact copy of the larger portions. In many fractals, the self-similarity is not exact, just as it is not exact in clouds. The Mandelbrot set (left), for example, contains many copies of itself. None of these copies is an exact duplicate, however. There are actually infinitely many copies of the entire set in the image below (the northernmost part of the Mandelbrot Set). If only I could give you infinite resolution. Anyway, most of the copies appear to be tiny black dots.

(b)
Light that reflects back at the same angle as the angle of incidence is known as "specular reflection light." The FTIR method that exploits specular reflection light is called the "specular reflection method." When using the specular reflection method, the sample can be irradiated with infrared light from an almost vertical angle of incidence or an almost horizontal angle of incidence. A specular reflection accessory that irradiates the sample with infrared light from an almost vertical angle is used to measure the infrared spectra of comparatively thick, μm-order, metal coatings or to measure the film thickness on epitaxial wafers using interference fringes.
However, when a specular reflection spectrum is measured for shiny samples that exhibit absorption in the infrared range, such as plastic, glass, or crystal, the peaks are deformed toward the first-order differential form due to the anomalous dispersion of the refractive

index in a range where absorption occurs. Qualification or functional group analysis is difficult from such a spectrum. Therefore, the deformed spectrum must be converted to a normal absorption spectrum using so-called Kramer's-Koenig transformation.

(1) Kramer's-Koenig Analysis

The explanation below applies to the irradiation of substance at an almost vertical angle of incidence. A more detailed explanation is given in the References [1], [2] below. The complex index of refraction of the substance n^* is set as follows:

$n^* = n + ik$ (where n and k are the refractive index and absorbance coefficient of the substance,respectively)

Then, the amplitude reflectance (r) under vertical irradiation and energy reflectance (R) determined directly from the specular reflection spectrum are expressed by the following equations:

$$r = \sqrt{R}e^{i\phi} = \sqrt{R}(\cos\phi + i\sin\phi) = \frac{n-ik-1}{n-ik+1} \quad (1)$$

$$R = |r|^2 = r \cdot r^* = \frac{(n-1)^2+k^2}{(n+1)^2+k^2} \quad (2)$$

Where, φ is the phase change. This results because both reflection and absorption occur at the sample surface. r^* is the complex conjugate of r. Dividing Equation (1) into real and imaginary parts and solving for n and k yields the following equations:

$$n = \frac{1-R}{1+R-2\sqrt{R}\cos\phi} \quad (3)$$

$$k = \frac{-2\sqrt{R}}{1+R-2\sqrt{R}\cos\phi} \quad (4)$$

Taking the logarithm of Equation (1), gives

$$\ln r = \ln\sqrt{R} + i\phi \quad (5)$$

However, $\sqrt{R}$ and φ are not independent, but linked by the Kramer's-Koenig relationship:

$$\phi(\nu_g) = \frac{2\nu_g}{\pi}\int_0^\infty \frac{\ln\sqrt{R(\nu)}}{\nu^2-\nu_g^2}\,d\nu \quad (6)$$

If the energy reflectance is measured across the entire wavenumber range, the phase change $\varphi(\nu_g)$ can be calculated at any required wavenumber ν_g using this relational expression, and then the optical constants n and k can be determined from Equations (3) and (4). Consequently, by calculating k at constant wavenumber intervals across the normal infrared range between 4600 and 400 cm-1 for example, an absorbance

coefficient spectrum equivalent to the transmission spectrum can be calculated from the specular reflection spectrum.

In the integration of Equation (6), a pole exists at $\nu = \nu_g$ but several methods have been proposed to handle this integration. Two typical methods are the Maclaurin method and the double Fourier transform method. In the Maclaurin method, the phase change $\varphi(\nu_g)$ is given by Equation (7)

$$\phi(\nu_g) = \frac{2\nu_g}{\pi} \times 2h \times \sum \frac{\ln\sqrt{R(\nu_j)}}{\nu_j^2 - \nu_g^2} \qquad (7)$$

Where, $h = \nu_{j+1} - \nu_j$

j = 2, 4, 6, ..., g-1, g+1, ... (if g is an odd number)
j = 1, 3, 5, ..., g-1, g+1, ... (if g is an even number)

The ν_j start point is set so that $\nu = \nu_g$ does not occur and alternate data points are used. With the double Fourier transform method, $\varphi(\nu_g)$ is determined by Equation (8), a double Fourier transformation which is an approximation of Equation (6).

$$\phi(\nu_g) = 4\int_0^\infty \cos(2\pi\nu_g t)\,dt \int_0^\infty \ln\sqrt{R(\nu)}\,\sin(2\pi\nu t)\,d\nu \qquad (8)$$

The Maclaurin method provides better calculation accuracy, but because it is so time-consuming, the quicker double Fourier transformation method is normally used.

(c)

Here, we introduce time interpolation to specify the animation paths between key frames and produce realistic displays of different speed changes, i.e., accelerations particularly at the beginning and at the end of a motion sequence. We now discuss curve fitting methods to determine the time spacing between frames using non linear interpolation.

Given the vertex positions at the key frames, we can fit the positions with linear or nonlinear paths. Figure 4 illustrates a non linear fit of key frame positions and this fit gives the trajectories for the in-betweens. But to simulate accelerations, we need to adjust the time spacing for the in-betweens. First, we consider constant speed (zero acceleration) using equal time-interval spacing for the in-betweens. Let there be n in-betweens for key frames at times t_1 and t_2 as shown in Fig. 5. We now divide the time interval between key frames into (n+1) sub intervals, yielding an in-between spacing of

$$\Delta t = \frac{t_2 - t_1}{n+1}$$

To produce realistic displays of speed changes particularly at the beginning and at the end of a motion sequence, we need to use non zero accelerations. For this, we can model the start-up and slow-down portions of an animation path with spline or trigonometric functions. Parabolic and cubic time functions could also be applied to model acceleration but trigonometric functions are more commonly used in animation packages.

To model an increasing speed (positive acceleration), the time spacing between frame has to be increased so that greater changes in position occur as the object moves faster. We can obtain an increasing interval size with the function

$1-\cos\theta \quad 0 < \theta < \pi/2$

The time for the jth in-between can be calculated from the above function as

$$t_{bj} = t_1 + \Delta t \left[1 - \cos\frac{j\pi}{2(n+1)}\right], \qquad j = 1,2,\ldots,n$$

Text Book

Computer Graphics with OpenGL by Hearn and Baker, Third Edition, 2009 (Indian Edition) Pearson Education