Computer Graphics Laboratory with Mini Project 17CSL68

Some Viva questions

1. What is the principle of Sierpinski gasket?
2. Difference between additive and subtractive color?
3. What is the Graphics Architecture used in OpenGL?
4. What is Rasterisation?
5. Explain sierpinski Gasket (using points).
6. Explain sierpinski Gasket (using polye)
7. Difference between 2D Tetrohedron and 3D Tetrahedron.
8. What do you mean by clipping?
9. Explain CohenSutherland Clipping Algorithm.
10. How do you set the color attributes in Opengl?
11. What is the command for clearscreen?
12. What is Event callback function?
13. Explain window to viewpoint mapping?
14. What are vertex arrays?
15. How are the faces of the color cube modeled?
16. How do you consider the inward and outward pointing of the faces?
17. Explain the OpenGL function used to rotate the color cube? What is Hidden surface Removal? How do you achieve this in OpenGL?
18. What are the functions for creating Menus in OpenGL?
19. Explain about fonts in GLUT?
20. What is the difference between 2D and 3D orthographic projection statements?

2l. Explain the Inward and Outward pointing face?

22. Explain the data structure for object representation?
23. Explain the vertex list representation of a cube?
24. What is transformation?
25. Explain the OpenGL functions used for translation, rotation and scaling?
26. What is the order of transformation?
27. State the difference between modelview and projection?
28. What is Homogeneous-coordinate representation?
29. Define the rotation matrix and object matrix.
30. Explain the procedure to obtain the resultant matrix using rotation matrix and object matrix.

3l. What is the principle of Cohen-Sutherland Algorithm?

32. State the advantages and disadvantages of Cohen-Sutherland Algorithm?
33. What is an outcode?
34. What is Synthetic Camera Model?
35. What are the Camera Specifications?
36. Explain the cases of outcodes in Cohen-Sutherland algorithm.

37. Explain the cohen-sutherland line clipping algorithm
38. Mention the difference between Liang-Barsky and Cohen-Sutherland line clipping algorithm.
39. Explain about gluLookAt(...), glFrustum(...), gluPerspective?
40. Explain the different types of projections?
4l. Explain Z-buffer algorithm?
42. What is antialiasing?
43. What is Center Of Projection(COP), Direction Of Projection(DOP)?
44. What is midpoint circle drawing algorithm
45. How do you get the equation d+=2x+3, d+=2(x-y)+5
46. What is gluOrtho2D function
47. Explain plot pixel function
48. Why do we use GLFlush function in Display
49. Explain Specular, Diffuse and Translucent surfaces.
50. What is ambient light? 5l. What is umbra, penumbra?
51. Explain Phong lighting model.
52. Explain glLightfv(...), glMaterialfv(...).
53. What is glutSolidCube function ? what are its Parameters
54. What are the parameters to glScale function
55. Explain Push & Pop matrix Functions
56. What is Materialfv function & its Parameters
57. Explain GLULookAt Function
58. Explain the keyboard and mouse events used in the program?
59. What is Hidden surface Removal? How do you achieve this in OpenGL?
60. Explain about glutPostRedisplay()?
61. Explain how the cube is constructed
62. Explain rotate function
63. What is GLFrustum function what are its Parameters
64. What is viewport
65. What is glutKeyboard Function what are its Parameters
66. Explain scanline filling algorithm?
67. What is AspectRatio,Viewport?
68. What are the different frames in OpenGL?
69. What is fragment processing?
70. Explain Polygon Filling algorithm
71. What is slope
72. How the edges of polygon are detected
73. Why you use GL PROJECTION in MatrixMode Function
74. What is glutPostRedisplay

75. Why do we use glutMainLoop function
76. What do you mean by GL LINE LOOP in GL Begin function
77. Define Computer Graphics.
78. Explain any 3 uses of computer graphics applications.
79. What are the advantages of DDA algorithm?
80. What are the disadvantages of DDA algorithm?
81. Define Scan-line Polygon fill algorithm.
82. What are Inside-Outside tests?
83. Define Boundary-Fill algorithm.
84. Define Flood-Fill algorithm.
85. What is a line width? What is the command used to draw the thickness of lines.
86. What are the three types of thick lines? Define.
87. What are the attribute commands for a line color?
88. What is color table? List the color codes.
89. What is a marker symbol and where it is used?
90. Discuss about inquiry functions.
91. Define translation and translation vector.
92. Define window and view port.
93. Define viewing transformation.
94. Give the equation for window to viewport transformation.
95. Define view up vector.
96. What is meant by clipping? Where it happens?
97. What is point clipping and what are its inequalities?
98. What is line clipping and what are their parametric representations?
99. How is translation applied?
100. What is referred to as rotation?
101. Write down the rotation equation and rotation matrix.
102. Write the matrix representation for scaling, translation and rotation.
103. Draw the block diagram for 2D viewing transformation pipeline.
104. Mention the equation for homogeneous transformation.
105. What is known as composition of matrix?
106. Write the composition transformation matrix for scaling, translation and Rotation.
107. Discuss about the general pivot point rotation?
108. Discuss about the general fixed point scaling.
109. Explain window, view port and window -to -view port transformation
110. Mention the three raster functions available in graphics packages.
111. What is known as region codes
112. Explain the three primary color used in graphics

113. Explain the area fill attributes and character attributes

114. Briefly discuss about basic transformations

115. Discuss about composite transformations.

116. Explain about reflection and shear

117. Explain the following transformation with the matrix representations. Give suitable diagram for illustration translation .ii scaling.iii rotation.

118. How the rotation of an object about the pivot point is performed?

Some Viva questions and answers

1. What is Computer Graphics? Answer: Computer graphics are graphics created using computers and, more generally, the representation and manipulation of image data by a computer.

2. What is OpenGL? Answer: OpenGL is the most extensively documented 3D graphics API (Application Program Interface) to date. It is used to create Graphics.

3. What is GLUT? Answer: The OpenGL Utility Toolkit (GLUT) is a library of utilities for OpenGL programs, which primarily perform system-level I/O with the host operating system.

4. What are the applications of Computer Graphics? Answer: Gaming Industry, Animation Industry and Medical Image Processing Industries. The sum total of these industries is a Multi Billion Dollar Market. Jobs will continue to increase in this arena in the future.

5. Explain in breif 3D Sierpinski gasket? Answer: The Sierpinski triangle (also with the original orthography Sierpinski), also called the Sierpinski gasket or the Sierpinski Sieve, is a fractal named after the Polish mathematician Waclaw Sierpinski who described it in 1915. Originally constructed as a curve, this is one of the basic examples of self-similar sets, i.e. it is a mathematically generated pattern that can be reproducible at any magnifcation or reduction.

6. What is Liang-Barsky line clipping algorithm? Answer: In computer graphics, the Liang-Barsky algorithm is a line clipping algorithm. The Liang-Barsky algorithm uses the parametric equation of a line and inequalities describing the range of the clipping box to determine the intersections between the line and the clipping box. With these intersections it knows which portion of the line should be drawn.

7. Explain in brief Cohen-Sutherland line-clipping algorithm? Answer: The Cohen-Sutherland line clipping algorithm quickly detects and dispenses with two common and trivial cases. To clip a line, we need to consider only its endpoints. If both endpoints of a line lie inside the window, the entire line lies inside the window. It is trivially accepted and needs no clipping. On the other hand, if both endpoints of a line lie entirely to one side of the window, the line must lie entirely outside of the window. It is trivially rejected and needs to be neither clipped nor displayed.

8. Explain in brief scan-line area flling algorithm? Answer: The scanline fll algorithm is an ingenious way of flling in irregular polygons. The algorithm begins with a set of points. Each point is connected to the next, and the line between them is considered to be an edge of the polygon. The points of each edge are adjusted to ensure that the point with the smaller y value appears frst. Next, a data structure is created that contains a list of edges that begin on each scanline of the image. The program progresses from the frst scanline upward. For each line, any pixels that contain an intersection between this scanline and an edge of the polygon are flled in. Then, the algorithm progresses along the scanline, turning on when it reaches a polygon pixel and turning of when it reaches another one, all the way across the scanline.

9. Explain Midpoint Line algorithm Answer: The Midpoint line algorithm is an algorithm which determines which points in an n-dimensional raster should be plotted in order to form a close approximation to a straight line between two given points. It is commonly used to draw lines on a computer screen, as it uses only integer addition, subtraction and bit shifting, all of which are very cheap operations in standard computer architectures.

10. What is a Pixel? Answer: In digital imaging, a pixel (or picture element) is a single point in a raster image. The Pixel is the smallest addressable screen element; it is the smallest unit of picture which can be controlled. Each Pixel has its address. The address of Pixels corresponds to its coordinate. Pixels are normally arranged in a 2-dimensional grid, and are often represented using dots or squares.

11. What is the general form of an OpenGL program? Answer: There are no hard and fast rules. The following pseudocode is generally recognized as good OpenGL form. program entrypoint { // Determine which depth or pixel format should be used. // Create a window with the desired format. // Create a rendering context and make it current with the window. // Set up initial OpenGL state. // Set up callback routines for window resize and window refresh. } handle resize { glViewport(...); glMatrixMode(GL PROJECTION); glLoadIdentity(); // Set projection transform with glOrtho, glFrustum, gluOrtho2D, gluPerspective, etc. } handle refresh { glClear(...); glMatrixMode(GL MODELVIEW); glLoadIdentity(); // Set view transform with gluLookAt or equivalent // For each object (i) in the scene that needs to be rendered: // Push relevant stacks, e.g., glPushMatrix, glPushAttrib. // Set OpenGL state specifc to object (i). // Set model transform for object (i) using glTranslatef, glScalef, glRotatef, and/or equivalent. // Issue rendering commands for object (i)./ Pop relevant stacks, (e.g., glPopMatrix, glPopAttrib.) // End for loop. // Swap bufers. }

12. What support for OpenGL does Open,Net,FreeBSD or Linux provide? Answer: The X Windows implementation, XFree86 4.0, includes support for OpenGL using Mesa or the OpenGL Sample Implementation. XFree86 is released under the XFree86 license. http://www.xfree86.org/

13. What is the AUX library? Answer: The AUX library was developed by SGI early in OpenGL's life to ease creation of small OpenGL demonstration programs. It's currently neither supported nor maintained. Developing OpenGL programs using AUX is strongly discouraged. Use the GLUT instead. It's more fexible and powerful and is available on a wide range of platforms. Very important: Don't use AUX. Use GLUT instead.

14. How does the camera work in OpenGL? Answer: As far as OpenGL is concerned, there is no camera. More specifcally, the camera is always located at the eye space coordinate (0., 0., 0.). To give the appearance of moving the camera, your OpenGL application must move the scene with the inverse of the camera transformation.

15. How do I implement a zoom operation? Answer: A simple method for zooming is to use a uniform scale on the ModelView matrix. However, this often results in clipping by the zNear and zFar clipping planes if the model is scaled too large. A better method is to restrict the width and height of the view volume in the Projection matrix.

16. What are OpenGL coordinate units? Answer: Depending on the contents of your geometry database, it may be convenient for your application to treat one OpenGL coordinate unit as being equal to one millimeter or one parsec or anything in between (or larger or smaller). OpenGL also lets you specify your geometry with coordinates of difering values. For example, you may fnd it convenient to model an airplane's controls in centimeters, its fuselage in meters, and a world to fy around in kilometers. OpenGL's ModelView matrix can then scale these diferent coordinate systems into the same eye coordinate space. It's the application's responsibility to ensure that the Projection and ModelView matrices are constructed to provide an image that keeps the viewer at an appropriate distance, with an appropriate feld of view, and keeps the zNear and zFar clipping planes at an appropriate range. An application that displays molecules in micron scale, for example, would probably not want to place the viewer at a distance of 10 feet with a 60 degree feld of view.

17. What is Microsoft Visual Studio? Answer: Microsoft Visual Studio is an integrated development environment (IDE) for developing windows applications. It is the most popular IDE for developing windows applications or windows based software.

18. What does the .gl or .GL fle format have to do with OpenGL? Answer: .gl fles have nothing to do with OpenGL, but are sometimes confused with it. .gl is a fle format for images, which has no relationship to OpenGL.

19. Who needs to license OpenGL? Who doesn't? Is OpenGL free software? Answer: Companies which will be creating or selling binaries of the OpenGL library will need to license OpenGL. Typical examples of

licensees include hardware vendors, such as Digital Equipment, and IBM who would distribute OpenGL with the system software on their workstations or PCs. Also, some software vendors, such as Portable Graphics and Template Graphics, have a business in creating and distributing versions of OpenGL, and they need to license OpenGL. Applications developers do NOT need to license OpenGL. If a developer wants to use OpenGL that developer needs to obtain copies of a linkable OpenGL library for a particular machine. Those OpenGL libraries may be bundled in with the development and/or run-time options or may be purchased from a third-party software vendor, without licensing the source code or use of the OpenGLtrademark.

20. How do we make shadows in OpenGL? Answer: There are no individual routines to control neither shadows nor an OpenGL state for shadows. However, code can be written to render shadows.

21. What is the use of Glutinit? Answer: void glutInit(int *argcp, char **argv); glutInit will initialize the GLUT library and negotiate a session with the window system. During this process, glutInit may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized.

22. Describe the usage of glutInitWindowSize and glutInitWindowPosition? Answer: void glutInitWindowSize(int width, int height); void glutInitWindowPosition(int x, int y); Windows created by glutCreateWindow will be requested to be created with the current initial window position and size. The intent of the initial window position and size values is to provide a suggestion to the window system for a window's initial size and position. The window system is not obligated to use this information. Therefore, GLUT programs should not assume the window was created at the specifed size or position. A GLUT program should use the window's reshape callback to determine the true size of the window.

23. Describe the usage of glutMainLoop? Answer: void glutMainLoop(void); glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

FOR YOUR KNOWLEDGE

INTRODUCTION

Computer graphics are graphics created using computers and, more generally, the representation and manipulation of image data by a computer hardware and software. The development of computer graphics, or simply referred to as CG, has made computers easier to interact with, and better for understanding and interpreting many types of data. Developments in computer graphics have had a profound impact on many types of media and have revolutionized the animation and video game industry. 2D computer graphics are digital images—mostly from two-dimensional models, such as 2D geometric models, text (vector array), and 2D data. 3D computer graphics in contrast to 2D computer graphics are graphics that use a three-dimensional representation of geometric data that is stored in the computer for the purposes of performing calculations and rendering images.

OPEN GL

OpenGL is the most extensively documented 3D graphics API(Application Program Interface) to date. Information regarding OpenGL is all over the Web and in print. It is impossible to exhaustively list all sources of OpenGL information. OpenGL programs are typically written in C and C++. One can also program OpenGL from Delphi (a Pascal-like language), Basic, Fortran, Ada, and other langauges. To compile and link OpenGL programs, one will need OpenGL header files. To run OpenGL programs one may need shared or dynamically loaded OpenGL libraries, or a vendor-specific OpenGL Installable Client Driver (ICD).

GLUT

The OpenGL Utility Toolkit (GLUT) is a library of utilities for OpenGL programs, which primarily perform system-level I/O with the host operating system. Functions performed include window definition, window control, and monitoring of keyboard and mouse input. Routines for drawing a number of geometric primitives (both in solid and wireframe mode) are also provided, including cubes, spheres, and cylinders. GLUT even has some limited support for creating pop-up menus. The two aims of GLUT are to allow the creation of rather portable code between operating systems (GLUT is cross-platform) and to make learning OpenGL easier. All GLUT functions start with the glut prefix (for example, glutPostRedisplay marks the current window as needing to be redrawn).

KEY STAGES IN THE OPENGL RENDERING PIPELINE:

- **Display Lists**

All data, whether it describes geometry or pixels, can be saved in a *display list* for current or later use. (The alternative to retaining data in a display list is processing the data immediately - also known as *immediate mode*.) When a display list is executed, the retained data is sent from the display list just as if it were sent by the application in immediate mode

- **Evaluators**

All geometric primitives are eventually described by vertices. Parametric curves and surfaces may be initially described by control points and polynomial functions called basis functions. Evaluators provide a method to derive the vertices used to represent the surface from the control points. The method is a polynomial mapping, which can produce surface normal, texture coordinates, colors, and spatial coordinate values from the control points.

- **Per-Vertex Operations**

For vertex data, next is the "per-vertex operations" stage, which converts the vertices into primitives. Some vertex data (for example, spatial coordinates) are transformed by 4 x 4 floating-point matrices. Spatial

coordinates are projected from a position in the 3D world to a position on your screen. If advanced features are enabled, this stage is even busier. If texturing is used, texture coordinates may be generated and transformed here. If lighting is enabled, the lighting calculations are performed using the transformed vertex, surface normal, light source position, material properties, and other lighting information to produce a color value.

- **Primitive Assembly**

Clipping, a major part of primitive assembly, is the elimination of portions of geometry which fall outside a half-space, defined by a plane. Point clipping simply passes or rejects vertices; line or polygon clipping can add additional vertices depending upon how the line or polygon is clipped. In some cases, this is followed by perspective division, which makes distant geometric objects appear smaller than closer objects. Then viewport and depth (z coordinate) operations are applied. If culling is enabled and the primitive is a polygon, it then may be rejected by a culling test. Depending upon the polygon mode, a polygon may be drawn as points or lines.

The results of this stage are complete geometric primitives, which are the transformed and clipped vertices with related color, depth, and sometimes texture-coordinate values and guidelines for the rasterization step.

- **Pixel Operations**

While geometric data takes one path through the OpenGL rendering pipeline, pixel data takes a different route. Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number of components. Next the data is scaled, biased, and processed by a pixel map. The results are clamped and then either written into texture memory or sent to the rasterization step If pixel data is read from the frame buffer, pixel-transfer operations (scale, bias, mapping, and clamping) are performed. Then these results are packed into an appropriate format and returned to an array in system memory.

There are special pixel copy operations to copy data in the framebuffer to other parts of the framebuffer or to the texture memory. A single pass is made through the pixel transfer operations before the data is written to the texture memory or back to the framebuffer.

- **Texture Assembly**

An OpenGL application may wish to apply texture images onto geometric objects to make them look more realistic. If several texture images are used, it’s wise to put them into texture objects so that you can easily switch among them.

Some OpenGL implementations may have special resources to accelerate texture performance. There may be specialized, high-performance texture memory. If this memory is available, the texture objects may be prioritized to control the use of this limited and valuable resource.

- **Rasterization**

Rasterization is the conversion of both geometric and pixel data into *fragments*. Each fragment square corresponds to a pixel in the framebuffer. Line and polygon stipples, line width, point size, shading model, and coverage calculations to support antialiasing are taken into consideration as vertices are connected into lines or the interior pixels are calculated for a filled polygon. Color and depth values are assigned for each fragment square.

- **Fragment Operations**

Before values are actually stored into the framebuffer, a series of operations are performed that may alter or even throw out fragments. All these operations can be enabled or disabled.
The first operation which may be encountered is texturing, where a texel (texture element) is generated from texture memory for each fragment and applied to the fragment. Then fog calculations may be applied, followed by the scissor test, the alpha test, the stencil test, and the depth-buffer test (the depth buffer is for hidden-surface removal). Failing an enabled test may end the continued processing of a fragment's square. Then, blending, dithering, logical operation, and masking by a bitmask may be performed.Finally, the thoroughly processedfragment is drawn into the appropriate buffer, where it has finally advanced to be a pixel and achieved its final resting place.

- **OpenGL-Related Libraries**

OpenGL provides a powerful but primitive set of rendering commands, and all higher-level drawing must be done in terms of these commands. Also, OpenGL programs have to use the underlying mechanisms of the windowing system. A number of libraries exist to allow you to simplify your programming tasks, including the following:

The OpenGL Utility Library (GLU) contains several routines that use lower-level OpenGL commands to perform such tasks as setting up matrices for specific viewing orientations and projections, performing polygon tessellation, and rendering surfaces. This library is provided as part of every OpenGL implementation. Portions of the GLU are described in the *OpenGL*

For every window system, there is a library that extends the functionality of that window system to support OpenGL rendering. For machines that use the X Window System, the OpenGL Extension to the X Window System (GLX) is provided as an adjunct to OpenGL. GLX routines use the prefix glX. For Microsoft

Windows, the WGL routines provide the Windows to OpenGL interface. All WGL routines use the prefix wgl. For IBM OS/2, the PGL is the Presentation Manager to OpenGL interface, and its routines use the prefix pgl.

The OpenGL Utility Toolkit (GLUT) is a window system-independent toolkit, written by Mark Kilgard, to hide the complexities of differing window system APIs. Open Inventor is an object-oriented toolkit based on OpenGL which provides objects and methods for creating interactive three-dimensional graphics applications. Open Inventor, which is written in C++, provides prebuilt objects and a built-in event model for user interaction, high-level application components for creating and editing three-dimensional scenes, and the ability to print objects and exchange data in other graphics formats. Open Inventor is separate from OpenGL.

- **GLUT, the OpenGL Utility Toolkit**

As you know, OpenGL contains rendering commands but is designed to be independent of any window system or operating system. Consequently, it contains no commands for opening windows or reading events from the keyboard or mouse. Unfortunately, it's impossible to write a complete graphics program without at least opening a window, and most interesting programs require a bit of user input or other services from the operating system or window system. In many cases, complete programs make the most interesting examples, so this book uses GLUT to simplify opening windows, detecting input, and so on. If you have an implementation of OpenGL and GLUT on your system, the examples in this book should run without change when linked with them.

In addition, since OpenGL drawing commands are limited to those that generate simple geometric primitives (points, lines, and polygons), GLUT includes several routines that create more complicated three-dimensional objects such as a sphere, a torus, and a teapot. This way, snapshots of program output can be interesting to look at. (Note that the OpenGL Utility Library, GLU, also has quadrics routines that create some of the same three-dimensional objects as GLUT, such as a sphere, cylinder, or cone.)

GLUT may not be satisfactory for full-featured OpenGL applications, but you may find it a useful starting point for learning OpenGL. The rest of this section briefly describes a small subset of GLUT routines so that you can follow the programming examples in the rest of this book.

OBJECTIVE AND APPLICATION OF THE LAB

The objective of this lab is to give students hands on learning exposure to understand and apply computer graphics with real world problems. The lab gives the direct experience to Visual Basic Integrated Development Environment (IDE) and GLUT toolkit. The students get a real world exposure to Windows programming API.

Applications of this lab are profoundly felt in gaming industry, animation industry and Medical Image Processing Industry. The materials learned here will useful in Programming at the Software Industry.

Setting up GLUT - main()

GLUT provides high-level utilities to simplify OpenGL programming, especially in interacting with the Operating System (such as creating a window, handling key and mouse inputs). The following GLUT functions were used in the above program:

- **glutInit:** initializes GLUT, must be called before other GL/GLUT functions. It takes the same arguments as the main().

  void **glutInit**(int *argc, char **argv)
- **glutCreateWindow:** creates a window with the given title.

  int **glutCreateWindow**(char *title)
- **glutInitWindowSize:** specifies the initial window width and height, in pixels.

  void **glutInitWindowSize**(int width, int height)
- **glutInitWindowPosition:** positions the top-left corner of the initial window at (x, y). The coordinates (x, y), in term of pixels, is measured in window coordinates, i.e., origin (0, 0) is at the top-left corner of the screen; x-axis pointing right and y-axis pointing down.

  void **glutInitWindowPosition**(int x, int y)
- **glutDisplayFunc:** registers the callback function (or event handler) for handling window-paint event. The OpenGL graphic system calls back this handler when it receives a window re-paint request. In the example, we register the function display() as the handler.

  void **glutDisplayFunc**(void (*func)(void))
- **glutMainLoop:** enters the infinite event-processing loop, i.e, put the OpenGL graphics system to wait for events (such as re-paint), and trigger respective event handlers (such as display()).

  void **glutMainLoop**()
- **glutInitDisplayMode:** requests a display with the specified mode, such as color mode (GLUT_RGB, GLUT_RGBA, GLUT_INDEX), single/double buffering (GLUT_SINGLE, GLUT_DOUBLE), enable depth (GLUT_DEPTH), joined with a bit OR '|'.

  void **glutInitDisplayMode**(unsigned int displayMode)
- **void glMatrixMode (GLenum mode);**

  The **glMatrixMode** function specifies which matrix is the current matrix.
- **void glOrtho(GLdoubleleft,GLdoubleright,GLdoublebottom,GLdoubletop,GLdoublezNear,GLdoublezFar);**

  The **glOrtho** function multiplies the current matrix by an orthographic matrix.

- **void glPointSize (GLfloat size);**

  The **glPointSize** function specifies the diameter of rasterized points.

- **voidglutPostRedisplay(void);**

  glutPostRedisplay marks the current window as needing to be redisplayed.

- **void glPushMatrix (void);**

  void glPopMatrix (void);

  The **glPushMatrix** and **glPopMatrix** functions push and pop the current matrix stack.

- **GLintglRenderMode (GLenum mode);**

  The **glRenderMode** function sets the rasterization mode.

- **void glRotatef (GLfloat angle, GLfloat x, GLfloat y, GLfloat);**

  The glRotatef functions multiply the current matrix by a rotation matrix.

- **void glScalef (GLfloat x, GLfloat y, GLfloat z);**

  The **glScalef** functions multiply the current matrix by a general scaling matrix.

- **void glTranslatef (GLfloat x, GLfloat y, GLfloat z);**

  The glTranslatef functions multiply the current matrix by a translation matrix.

- **void glViewport (GLint x, GLint y, GLsizei width, GLsizei height);**

  The **glViewport** function sets the viewport.

- **void glEnable, glDisable();**

  The **glEnable** and **glDisable** functions enable or disable OpenGL capabilities.

- **glutBitmapCharacter();**

  The **glutBitmapCharacter** function used for font style.

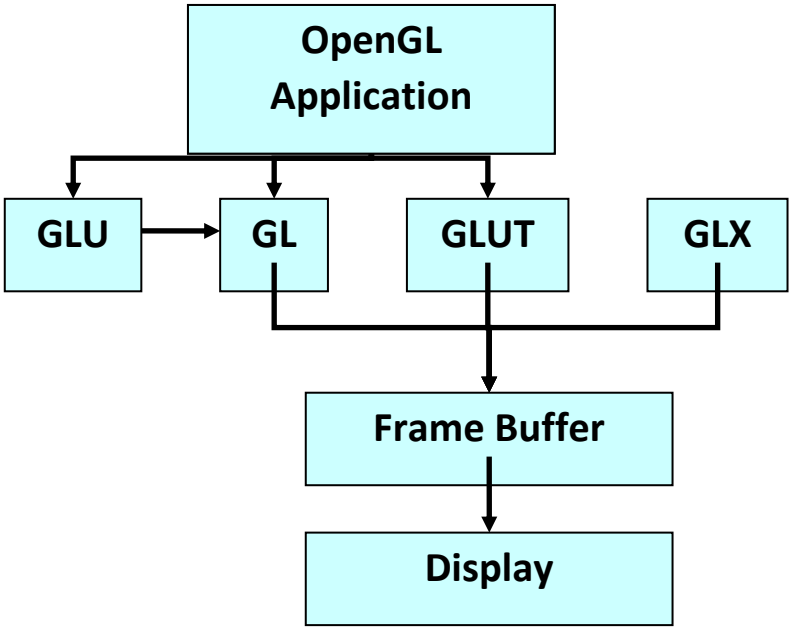
OpenGL Primitives

Value	Meaning
GL_POINTS	individual points
GL_LINES	pairs of vertices interpreted as individual line segments
GL_POLYGON	boundary of a simple, convex polygon
GL_TRIANGLES	triples of vertices interpreted as triangles
GL_QUADS	quadruples of vertices interpreted as four-sided polygons
GL_LINE_STRIP	series of connected line segments
GL_LINE_LOOP	same as above, with a segment added between last and first vertices
GL_TRIANGLE_STRIP	linked strip of triangles
GL_TRIANGLE_FAN	linked fan of triangles
GL_QUAD_STRIP	linked strip of quadrilaterals

INTRODUCTION TO OpenGL

OpenGL is an API.OpenGL is nothing more than a set of functions you call from your program (think of as collection of .h files.

OpenGL Libraries:



OpenGL Hierarchy:

- Several levels of abstraction are provided
- GL
  - Lowest level: vertex, matrix manipulation
  - glVertex3f(point.x, point.y, point.z)
- GLU
  - Helper functions for shapes, transformations
  - gluPerspective(fovy, aspect, near, far)
  - gluLookAt(0, 0, 10, 0, 0, 0, 0, 1, 0);
- GLUT
  - Highest level: Window and interface management
  - glutSwapBuffers()
  - glutInitWindowSize (500, 500);

OpenGL Implementations :

- OpenGL IS an API (think of as collection of .h files):
  - #include <GL/gl.h>
  - #include <GL/glu.h>
  - #include <GL/glut.h>
- Windows, Linux, UNIX, etc. all provide a platform specific implementation.
- Windows: opengl32.lib glu32.lib glut32.lib
- Linux: -l GL -l GLU –l GLUT

Event Loop:

- OpenGL programs often run in an event loop:
  - Start the program
  - Run some initialization code
  - Run an infinite loop and wait for events such as
    - Key press
    - Mouse move, click
    - Reshape window
    - Expose event

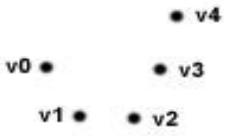
OpenGL Command Syntax (1) :

- OpenGL commands start with “gl”
- OpenGL constants start with “GL_”
- Some commands end in a number and one, two or three letters at the end (indicating number and type of arguments)
- A Number indicates number of arguments
- Characters indicate type of argument

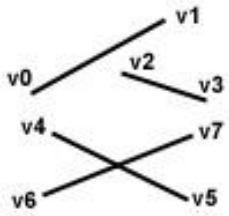
OpenGL Command Syntax (2)

- `f' float
- `d' double float
- `s' signed short integer
- `i' signed integer
- `b' character
- `ub' unsigned character
- `us' unsigned short integer
- `ui' unsigned integer

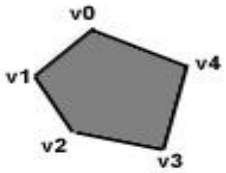
Ten gl Primitives:



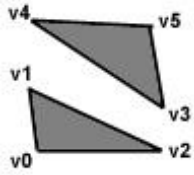
GL_POINTS



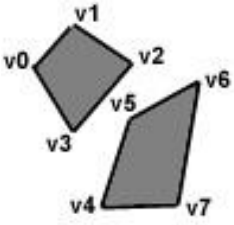
GL_LINES



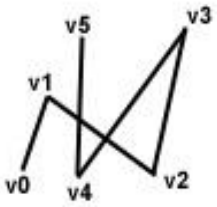
GL_POLYGON



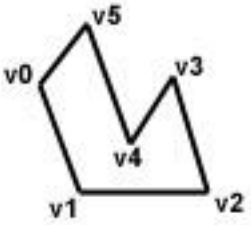
GL_TRIANGLES



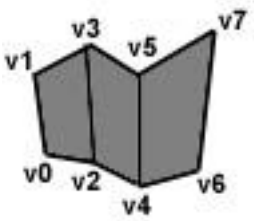
GL_QUADS



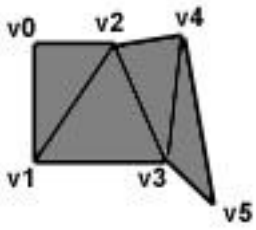
GL_LINE_STRIP



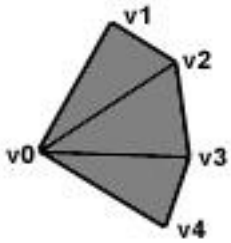
GL_LINE_LOOP



GL_QUAD_STRIP



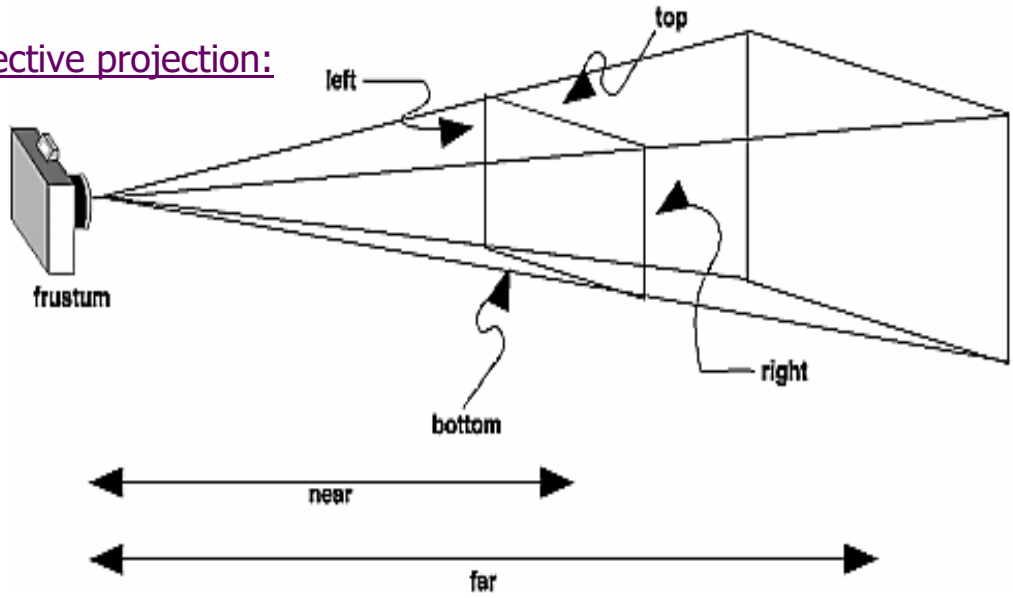
GL_TRIANGLE_STRIP



GL_TRIANGLE_FAN

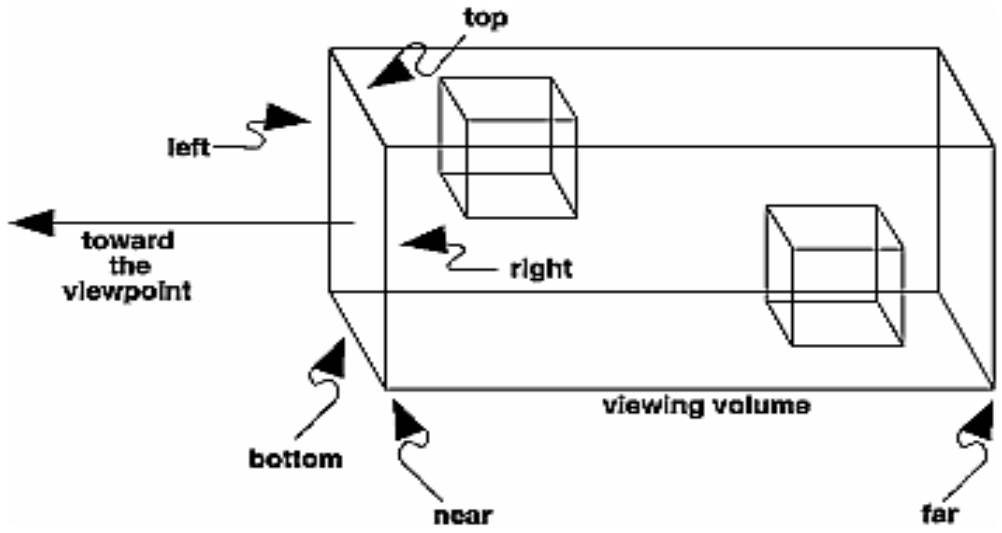
Projections in OpenGL

Perspective projection:



*void **glFrustum**(GLdouble left, GLdouble right, GLdouble bottom,GLdouble top, GLdouble near, GLdouble far);*

Orthographic projection:



*void **glOrtho**(GLdouble left, GLdouble right, GLdouble bottom,*

GLdouble top, GLdouble near, GLdouble far)

Introduction to frequently used OpenGL commands:

glutInit

`glutInit` is used to initialize the GLUT library.

Usage

void glutInit(int *argcp, char **argv);

argcp

A pointer to the program's *unmodified*argc variable from main. Upon return, the value pointed to by argcp will be updated, because glutInit extracts any command line options intended for the GLUT library.

argv

The program's *unmodified*argv variable from main. Like argcp, the data for argv will be updated because glutInit extracts any command line options understood by the GLUT library.

Description

`glutInit` will initialize the GLUT library. During this process, `glutInit` may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized.

glutInitWindowPosition, glutInitWindowSize

`glutInitWindowPosition` and `glutInitWindowSize` set the *initial window position* and *size* respectively.

Usage

void glutInitWindowSize(int width, int height);

void glutInitWindowPosition(int x, int y);

width

Width in pixels.

height

Height in pixels.

x

Window X location in pixels.

y

Window Y location in pixels.

Description

Windows created by `glutCreateWindow` will be requested to be created with the current *initial window position* and *size*.

glutInitDisplayMode

`glutInitDisplayMode` sets the *initial display mode*.

Usage

void glutInitDisplayMode(unsigned int mode);

mode

Display mode, normally the bitwise *OR*-ing of GLUT display mode bit masks. See values below:

GLUT_RGB

An alias for GLUT_RGBA.

GLUT_INDEX

Bit mask to select a color index mode window. This overrides GLUT_RGBA if it is also specified.

GLUT_SINGLE

Bit mask to select a single buffered window. This is the default if neither GLUT_DOUBLE or GLUT_SINGLE are specified.

GLUT_DOUBLE

Bit mask to select a double buffered window. This overrides GLUT_SINGLE if it is also specified.

GLUT_DEPTH

Bit mask to select a window with a depth buffer.

Description

The *initial display mode* is used when creating top-level windows.

glutMainLoop

`glutMainLoop` enters the GLUT event processing loop.

Usage

void glutMainLoop(void);

Description

`glutMainLoop` enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

glutCreateWindow

`glutCreateWindow` creates a top-level window.

Usage

int glutCreateWindow(char *name);

name

ASCII character string for use as window name.

Description`glutCreateWindow` creates a top-level window. The `name` will be provided to the window system as the window's name. The intent is that the window system will label the window with the name.

glutPostRedisplay

`glutPostRedisplay` marks the *current window* as needing to be redisplayed.

Usage

void glutPostRedisplay(void);

Description

Mark the normal plane of *current window* as needing to be redisplayed. The next iteration through `glutMainLoop`, the window's display callback will be called to redisplay the window's normal plane. Multiple calls to `glutPostRedisplay` before the next display callback opportunity generates only a single redisplay callback.

glutReshapeWindow

`glutReshapeWindow` requests a change to the size of the *current window*.

Usage

void glutReshapeWindow(int width, int height);

width

New width of window in pixels.

height

New height of window in pixels.

Description

`glutReshapeWindow` requests a change in the size of the *current window*. The `width` and `height` parameters are size extents in pixels. The `width` and `height` must be positive values.

The requests by `glutReshapeWindow` are not processed immediately. The request is executed after returning to the main event loop. This allows multiple `glutReshapeWindow`, `glutPositionWindow`, and `glutFullScreen` requests to the same window to be coalesced.

glutDisplayFunc

`glutDisplayFunc` sets the display callback for the *current window*.

Usage

void glutDisplayFunc(void (*func)(void));

func

The new display callback function.

Description

`glutDisplayFunc` sets the display callback for the *current window*. When GLUT determines that the normal plane for the window needs to be redisplayed, the display callback for the window is called. Before the callback, the *current window* is set to the window needing to be redisplayed and (if no overlay display callback is registered) the *layer in use* is set to the normal plane. The display callback is called with no parameters. The entire normal plane region should be redisplayed in response to the callback (this includes ancillary buffers if your program depends on their state).

glutReshapeFunc

`glutReshapeFunc` sets the reshape callback for the *current window*.

Usage

void glutReshapeFunc(void (*func)(int width, int height));

func

The new reshape callback function.

Description

`glutReshapeFunc` sets the reshape callback for the *current window*. The reshape callback is triggered when a window is reshaped. A reshape callback is also triggered immediately before a window's first display callback after a window is created or whenever an overlay for the window is established. The `width` and `height` parameters of the callback specify the new window size in pixels. Before the callback, the *current window* is set to the window that has been reshaped.

glFlush

Name

glFlush - force execution of GL commands in finite time

Description

Different GL implementations buffer commands in several different locations, including network buffers and the graphics accelerator itself. glFlush empties all of these buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in finite time.

glMatrixMode

Name

glMatrixMode - specify which matrix is the current matrix

Usage

void glMatrixMode(GLenum mode)

Parameters

mode- Specifies which matrix stack is the target for subsequent matrix operations. Three values are accepted:GL_MODELVIEW, GL_PROJECTION, and GL_TEXTURE. The default value is GL_MODELVIEW.

Description

glMatrixMode sets the current matrix mode. mode can assume one of three values:

GL_MODELVIEW - Applies subsequent matrix operations to the mode matrix stack.

GL_PROJECTION- Applies subsequent matrix operations to the projection matrix stack.

gluOrtho2D

Usage

gluOrtho2D(left, right, bottom, top)

- Specifies the 2D region to be projected into the viewport.
- Any drawing outside the region will be automatically clipped away.