Chapter 9 (6 Marks)

12. Discuss the need of OpenGL. Explain callback function. [4]

***

[2+4+2]

10. Write the importance of OpenGL in computer graphics. Write OpenGL syntax to draw a rectangle and polygon considering your own vertices. [6]

***

Explain briefly different ways of shading this triangle. [8+6]

9. Why OpenGL is used? Write the basic command to draw the pixel rectangle and polygon in OpenGL. [2+4]

***

[2+6]

10. Write down the Open GL syntax to draw basic 2D geometric primitives with examples. [5]

***

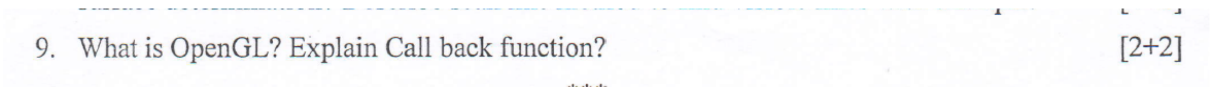
.0. What is OpenGL? How can we draw colored line and polygon using OpenGL? [2+4]

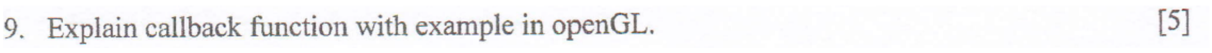
***

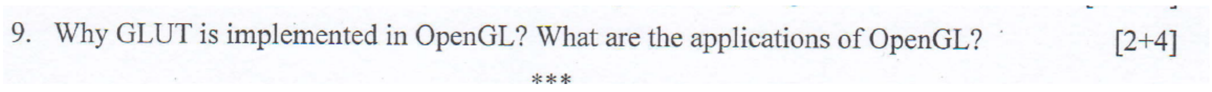
10. What do you mean call back function? Illustrate with example. [4]

***

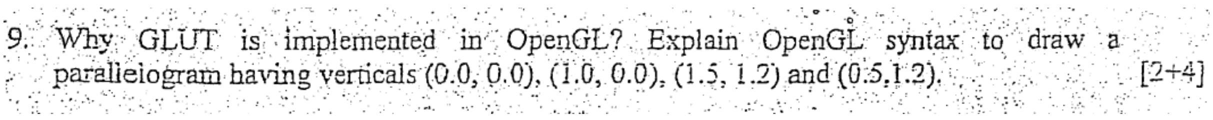
8. How polygon is drawn in OpenGL? How lighting is applied to this polygon surface? [2+3]

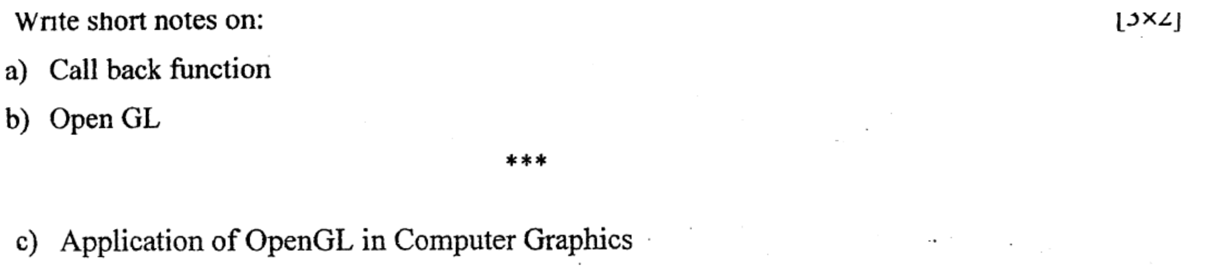

9. What is OpenGL? Explain Call back function? [2+2]


9. Explain callback function with example in openGL. [5]


9. Why GLUT is implemented in OpenGL? What are the applications of OpenGL? [2+4]

***


9. Why GLUT is implemented in OpenGL? Explain OpenGL syntax to draw a parallelogram having verticals (0.0, 0.0), (1.0, 0.0), (1.5, 1.2) and (0.5,1.2). [2+4]


Write short notes on: [5×2]

a) Call back function

b) Open GL

***

c) Application of OpenGL in Computer Graphics

OpenGL

OpenGL (Open Graphics Library) is a widely used cross-platform API for rendering 2D and 3D vector graphics. It allows developers to create visually rich applications, such as games, simulations, and CAD programs, by providing a set of functions to interact with a computer's GPU. OpenGL is hardware-accelerated, meaning it can efficiently render complex scenes and effects in real-time. It supports various features, including texture mapping, shading, and geometric transformations. OpenGL is highly portable, running on various operating systems like Windows, macOS, and Linux, and is used in both desktop and mobile applications.

Application in computer graphics

1. **Video Games:**
    - OpenGL is extensively used in the development of video games, allowing for the rendering of complex 3D environments, characters, and special effects.
2. **Virtual Reality (VR) and Augmented Reality (AR):**
    - In VR and AR applications, OpenGL helps create immersive environments by rendering 3D scenes in real-time.
3. **Computer-Aided Design (CAD):**
    - OpenGL is used in CAD software to visualize and manipulate 3D models of buildings, machinery, and other complex structures.
4. **Medical Imaging:**
    - OpenGL is applied in medical imaging systems to render 3D visualizations of organs and tissues from MRI, CT scans, and other imaging technologies.
5. **Animation and Film:**
    - OpenGL is used in the animation and film industry to create and render 3D scenes and effects.
6. **Educational Software:**
    - Educational applications use OpenGL to create interactive and visual learning tools.
7. **Graphical User Interfaces (GUIs):**
    - OpenGL is used to create advanced GUIs with 3D elements, smooth transitions, and animations.

Importance of OpenGL in computer graphics

- **Cross-Platform Compatibility**: Runs on multiple operating systems without code changes.

- **Industry Standard**: Widely adopted and supported by major hardware and software.
- **High Performance**: Provides direct access to the GPU for efficient rendering.
- **Rich Feature Set**: Offers advanced graphics techniques like shading and texture mapping.
- **Wide Adoption**: Used in popular applications and game engines.
- **Open Source Community**: Strong community support with extensive resources.

GLUT

GLUT (OpenGL Utility Toolkit) is a library of utilities for OpenGL programs that provides a framework for developing OpenGL applications. It is designed to be platform-independent, allowing developers to create OpenGL applications that can run on different operating systems without needing to modify the code. GLUT simplifies the process of managing windows, handling user input, and creating basic user interfaces, such as menus and buttons.

Key Features of GLUT:

- **Window Management**: GLUT manages the creation and handling of windows, making it easier for developers to focus on OpenGL rendering.
- **Event Handling**: It provides mechanisms for handling input events like keyboard and mouse interactions.
- **Timers and Idle Callbacks**: GLUT offers functions to set timers and idle callbacks, which are useful for animation and updating the screen.
- **Utility Functions**: GLUT includes functions to draw basic shapes, manage fonts, and more, simplifying the development process.

Why is GLUT Used in OpenGL?

GLUT was implemented to provide a simple and portable way to create OpenGL applications. It abstracts away the complexities of interacting with the underlying operating system, such as window management and input handling, which can vary significantly between platforms. By using GLUT, developers can write OpenGL code once and run it on multiple platforms with minimal changes.

OpenGL Callback Functions

Callback functions in OpenGL are a way to handle specific events or situations during the execution of a program. Instead of continuously checking for these events (like input from a user or the completion of a drawing operation), OpenGL allows you to define functions (callbacks) that automatically get called when these events occur. This mechanism makes the code cleaner, more modular, and easier to manage.

1. **Display Callback**: This function is responsible for rendering the scene. It's called whenever the window needs to be redrawn, such as when it is resized.

   ```
   void display() {
       glClear(GL_COLOR_BUFFER_BIT);
       // Render your objects here
       glFlush();
   }
   ```

   You register this callback using:

   ```
   glutDisplayFunc(display);
   ```

2. **Idle Callback**: This function is called when there are no other events to process. It's often used for animations or to update the scene continuously.

   ```
   void idle() {
       // Update animation or scene
       glutPostRedisplay(); // Request to redraw the window
   }
   ```

   Register it with:

   ```
   glutIdleFunc(idle);
   ```

3. **Keyboard Callback**: This function handles keyboard input. It is triggered whenever a key is pressed.

   ```
   void keyboard(unsigned char key, int x, int y) {
       switch (key) {
           case 'q': // Example key 'q'
               exit(0); // Exit the program
               break;
           // Handle other keys here
       }
   }
   ```

   Register it with:

   ```
   glutKeyboardFunc(keyboard);
   ```

4. **Special Keyboard Callback**: This function is similar to the keyboard callback but handles special keys like arrow keys or function keys.

   ```
   void special(int key, int x, int y) {
       switch (key) {
           case GLUT_KEY_LEFT:
               // Handle left arrow key
               break;
           case GLUT_KEY_RIGHT:
               // Handle right arrow key
               break;
           // Handle other special keys here
       }
   }
   ```

   Register it with:

   ```
   glutSpecialFunc(special);
   ```

5. **Mouse Callback**: This function manages mouse events such as clicks and movements.

```
void mouse(int button, int state, int x, int y) {
   if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
      // Handle left mouse button press
   }
}
```

Register it with:

```
glutMouseFunc(mouse);
```

In summary, OpenGL callback functions are integral for creating interactive and dynamic applications. They allow you to manage rendering, handle user input, and respond to window events effectively.

To draw basic 2D primitives in OpenGL

1. Point

To draw a single point:

```
glColor3f(1.0, 0.0, 0.0); // Set the color to red
glBegin(GL_POINTS);      // Begin drawing points
glVertex3f(0.5, 0.5, 0.0); // Specify the position of the point
glEnd();                // End drawing
```

2. Line

To draw a line between two points:

```
glColor3f(0.0, 1.0, 0.0); // Set the color to green
glBegin(GL_LINES);       // Begin drawing lines
glVertex3f(0.25, 0.25, 0.0); // Specify the first point
glVertex3f(0.75, 0.75, 0.0); // Specify the second point
glEnd();                // End drawing
```

3. Triangle

To draw a triangle:

```
glColor3f(0.0, 0.0, 1.0); // Set the color to blue
glBegin(GL_TRIANGLES);     // Begin drawing a triangle
glVertex3f(0.25, 0.25, 0.0); // Specify the first vertex
glVertex3f(0.75, 0.25, 0.0); // Specify the second vertex
glVertex3f(0.5, 0.75, 0.0);  // Specify the third vertex
glEnd();                   // End drawing
```

4. Rectangle (Quad)

To draw a rectangle, you can use a quadrilateral (quad):

```
glColor3f(1.0, 1.0, 0.0); // Set the color to yellow
glBegin(GL_QUADS);        // Begin drawing a quad (rectangle)
glVertex3f(0.25, 0.25, 0.0); // Specify the first vertex
glVertex3f(0.75, 0.25, 0.0); // Specify the second vertex
glVertex3f(0.75, 0.75, 0.0); // Specify the third vertex
glVertex3f(0.25, 0.75, 0.0); // Specify the fourth vertex
glEnd();                   // End drawing
```

5. Polygon

For more complex shapes, you can use `GL_POLYGON`:

```
glColor3f(0.0, 1.0, 1.0); // Set the color to cyan
glBegin(GL_POLYGON);      // Begin drawing a polygon
glVertex3f(0.25, 0.25, 0.0); // Specify vertices
glVertex3f(0.75, 0.25, 0.0);
glVertex3f(0.85, 0.50, 0.0);
glVertex3f(0.75, 0.75, 0.0);
glVertex3f(0.25, 0.75, 0.0);
glVertex3f(0.15, 0.50, 0.0);
glEnd();                   // End drawing
```

Notes:

- **`glBegin()` and `glEnd()`** are used to specify the start and end of the shape you want to draw.

- The argument passed to `glBegin()` specifies the type of primitive to draw (e.g., `GL_POINTS`, `GL_LINES`, `GL_TRIANGLES`, `GL_QUADS`, `GL_POLYGON`).
- `glVertex3f(x, y, z)` specifies a vertex in 3D space. The `z` coordinate is `0.0` for 2D drawing.

You can use these basic templates to draw various 2D shapes in OpenGL.

```
cppCopy code
glColor3f(0.0, 1.0, 1.0);  // Set the color to cyan
glBegin(GL_POLYGON);       // Begin drawing a polygon
glVertex2f(0.25, 0.25);    // Specify vertices
glVertex2f(0.75, 0.25);
glVertex2f(0.85, 0.50);
glVertex2f(0.75, 0.75);
glVertex2f(0.25, 0.75);
glVertex2f(0.15, 0.50);
glEnd();                   // End drawing
```

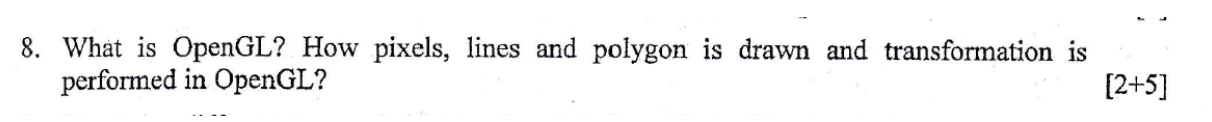
Summary:

- `glVertex2f(x, y)` is used to specify vertices with only `x` and `y` coordinates, ideal for 2D drawing.

To draw a parallelogram using the vertices provided in the image `(0.0, 0.0)`, `(1.0, 0.0)`, `(1.5, 1.2)`, and `(0.5, 1.2)` in OpenGL, you can use the following code snippet. Since these vertices define a quadrilateral, you can use the `GL_QUADS` primitive.

Code:

```
glColor3f(0.5, 0.5, 1.0);  // Set the color to a light blue
glBegin(GL_QUADS);         // Begin drawing a quadrilateral (parallelogram)
glVertex2f(0.0, 0.0);      // First vertex
glVertex2f(1.0, 0.0);      // Second vertex
glVertex2f(1.5, 1.2);      // Third vertex
glVertex2f(0.5, 1.2);      // Fourth vertex
glEnd();                   // End drawing
```


8. What is OpenGL? How pixels, lines and polygon is drawn and transformation is performed in OpenGL? [2+5]

In OpenGL, transformations like translation, rotation, and scaling are performed using transformation matrices. These transformations are applied to the vertices of your objects to move, rotate, or resize them in the 2D or 3D space. Here's how each transformation can be performed:

1. Translation (Moving the Object)

Example: Translating a parallelogram

```
glPushMatrix();           // Save the current matrix
glTranslatef(2, 2, 0.0); // Translate the object by (2, 2)
glBegin(GL_QUADS);        // Begin drawing the parallelogram
glVertex2f(0.0, 0.0);
glVertex2f(1.0, 0.0);
glVertex2f(1.5, 1.2);
glVertex2f(0.5, 1.2);
glEnd();                  // End drawing
glPopMatrix();            // Restore the previous matrix
```

2. Rotation

Rotation turns an object around a specific point, typically the origin, by a given angle.

Example: Rotating a parallelogram by 45 degrees around the origin

```
glPushMatrix();              // Save the current matrix
glRotatef(45.0, 0.0, 0.0, 1.0); // Rotate by 45 degrees around the z-axis
glBegin(GL_QUADS);           // Begin drawing the parallelogram
glVertex2f(0.0, 0.0);
glVertex2f(1.0, 0.0);
```

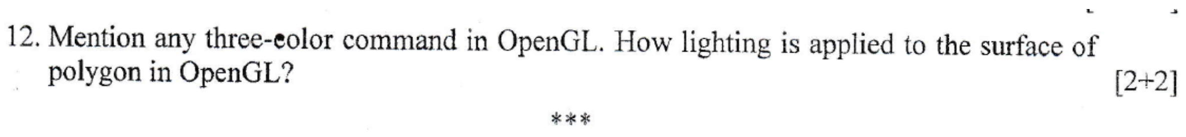
```
glVertex2f(1.5, 1.2);
glVertex2f(0.5, 1.2);
glEnd();                  // End drawing
glPopMatrix();              // Restore the previous matrix
```

3. Scaling

Scaling changes the size of an object by multiplying the coordinates of the vertices by scaling factors.

Example: Scaling a parallelogram by a factor of 2 in both x and y directions

```
glPushMatrix();          // Save the current matrix
glScalef(2.0, 2.0, 1.0);   // Scale the object by 2 in x and y directions
glBegin(GL_QUADS);        // Begin drawing the parallelogram
glVertex2f(0.0, 0.0);
glVertex2f(1.0, 0.0);
glVertex2f(1.5, 1.2);
glVertex2f(0.5, 1.2);
glEnd();                // End drawing
glPopMatrix();            // Restore the previous matrix
```



12. Mention any three-color command in OpenGL. How lighting is applied to the surface of polygon in OpenGL? [2+2]

***

Sure! Here are three OpenGL color commands:

1. `glColor3f(r, g, b)` : Sets the current color using floating-point values for red, green, and blue components.
2. `glColor4f(r, g, b, a)` : Sets the current color with red, green, blue, and alpha (transparency) components.
3. `glColor3ub(r, g, b)` : Sets the current color using unsigned byte values (0 to 255) for red, green, and blue components.

In OpenGL, lighting on polygons is done using the Phong model, which calculates how light interacts with the surface based on ambient, diffuse, and specular effects. Normals at each vertex help determine how light interacts with the surface. The color for each vertex is computed based on these effects, and then the color is smoothly blended across the entire polygon.