Assignment 4

1. What are the common features of every graphical file format? Explain any one of the Graphics file formats that you are familiar with.

→ Common features of graphical file formats:

1. Header Information

→ Every graphical file format contains a header that includes essential details about the file, such as its format, dimensions, color depth, compression method (if any), and other metadata.

2. Pixel Data

→ Graphical file formats store information about the color of each pixel in the image.

→ This data is arranged in a way that preserves the spatial information of the image.

3. Color Model

→ A color model defines how colors are represented & stored in the file.

→ Common color models include RGB (Red, Green, Blue), CMYK (Cyan, Magenta, Yellow, Key (Black), and Grayscale.

4. Compression
→ Many graphical file formats incorporate compression algorithms to reduce the file size without significant loss of image quality.
→ Different formats use various compression methods.

5. Transparency
→ Some formats support transparency, allowing certain parts of the image to be transparent, revealing the background or underlying layers when displayed or overlaid.

6. Metadata
→ Graphics file formats often include metadata, which can include information about the image, such as the creator, creation date, copyright details, & more

One example of a graphical file format is JPEG (Joint Photographic Experts Group):

→ JPEG is a commonly used lossy compression format for digital images.
→ It is widely used for photographs & images with complex color gradients.
→ The JPEG format allows for a good balance between image quality & file size, making it suitable for online sharing & storage.

Here's how JPEG works:

- **Compression**

→ JPEG achieves compression by reducing the amount of data stored for each image.

- **Color Model**

→ JPEG primarily uses the RGB color model to represent images.

→ Each pixel is represented by three color channels: red, green, and blue, with varying intensities to create a wide range of colors.

- **Header Information**

→ The JPEG file format begins with a header that contains essential information about the image, such as its format, dimensions, color space, & other metadata.

- **Subsampling**

→ JPEG supports chroma subsampling, which reduces the amount of color information for certain pixels, further reducing file size while preserving image quality. The most common subsampling ratios are 4:4:2, 4:2:2, and 4:2:0

- Compatibility

→ JPEG is widely supported by web browsers, image viewers, & editing slw, making it a popular choice for sharing & displaying images on the internet.

Q Why is it required to follow Graphics Standards? Explain any one of the Graphics standards that is used for display purpose. What is Language Binding?

→ Following graphics standards is essential for ensuring compatibility, interoperability, & consistency across different platforms & applications.

→ Graphics standards define a set of rules and specifications that slw developers & hlw manufactures adhere to when creating graphics-related slw or devices.

→ By following these standards, developers can ensure that their products work seamlessly with other compliant systems & avoid potential issues with file formats, rendering, & performance.

One widly used graphics standard for display purposes is VESA Display Data channel (DDC):

VESA Display Data channel (DDC):

→ The VESA Display Data channel (DDC) standard is an extension of the video Electronics Standards Association (VESA) Display Monitor Timing (DMT) standard.

→ It allows for two-way communication between a computer's graphics adapter & the monitor, enabling the graphics adapter to obtain essential information about the monitor's capabilities & configuration.

Here's how DDC works:

1. EDID (Extended Display Identification Data):

→ DDC relies on the EDID data structure, which is stored in the monitor's EEPROM.

→ The EDID contains detailed information about the monitor, such as its supported display resolutions, refresh rates, color capabilities, & other monitor-specific data.

2. Communication Protocol

→ When a computer is connected to a monitor with DDC support, the graphics adapter sends a query to the monitor requesting the EDID information.

→ The monitor responds by transmitting its EDID data back to the graphics adapter.

3. Automatic Configuration

→ With the EDID data, the graphics adapter can automatically configure the display settings to match the monitor's capabilities.

→ This ensures that the computer sends compatible video signals to the monitor, resulting in optimal image quality & preventing issues like incorrect aspect ratios or unsupported resolutions.

Language Binding:

→ Language binding, in the context of graphics programming, refers to the process of creating interfaces that allow a programming language to interact with a library or API (Application Programming Interface) written in another language.

→ The purpose of language binding is to enable developers to use graphics-related functionalities from different programming languages seamlessly.

For eg, if there is a graphics library written in C or C++, a language binding may be created to allow developers using languages like Python, Java, or C# to utilize the features of the graphics library without having to rewrite the entire library in their language of choice.

Q) Explain the need for machine independent graphical languages.

→ Machine-independent graphical languages are essential for several reasons:

1. Platform Independence

→ Different computer systems & architectures vary in terms of hlw, OSs, slw environments.

→ A machine-independent graphical language allows graphics to be created & rendered consistently across various platforms, ensuring that the same graphical content looks & behaves the same way regardless of the underlying hlw or OS.

2. Cross-Platform Compatibility

→ In todays diverse computing landscape, applications & content are accessed on a wide range of devices, including desktop computers, laptops, tablets, smartphones, and other embedded systems.

→ A machine-independent graphical language enables graphics to be displayed on different devices without requiring modifications or adaptions for each specific platform.

3. Ease of Development
→ Machine-independent graphical languages abstract the complexities of low-level hardware & platform-specific graphics APIs.
→ This simplifies the development process, as developers can create graphics using a single set of instructions or codebase, rather than having to write platform-specific code for each target system.

4. Interoperability
→ Machine-independent graphical languages promote interoperability between different software tools & applications.
→ It allows graphics to be easily exchanged & shared across various software packages without losing fidelity or encountering compatibility issues.

5. Scalability
→ Graphics designed using machine-independent languages can be easily scaled to different resolutions & screen sizes.
→ This is crucial for responsive design & adaptability to various display devices, from small mobile screens to large high-resolution monitors.

6. Reduced Development Time & Cost.
7. Future-Proofing
8. Standardization

Q4) Explain the architecture of OpenGL along with the API's used for graphical rendering of objects.

→ OpenGL (Open Graphics Library) is an open-source cross-platform graphics API (Application Programming Interface) used for rendering 2D and 3D graphics.

→ It provides a set of functions a set of functions that allow developers to interact with the GPU (Graphics Processing Unit) & create interactive & visually appealing graphics applications.

→ OpenGL follows a client-server architecture, where the client represents the application and the server represents the GPU.

1) Client (Application) Side:

→ The application or client side of OpenGL consists of the program or s/w that uses the OpenGL API to create & manage graphics objects & renders them on the screen.

→ The client side provides commands to OpenGL thru its API to perform various graphics operations, such as setting up the rendering pipeline, creating & manipulating geometric objects, applying transformations, & specifying rendering attributes.

→ OpenGL is platform-independent, so the same OpenGL code can be used on different OSs & hlw, making it highly portable.

2) Server (GPU) side

→ The server side of openGL is represented by the GPU (Graphics Processing Unit) which is responsible for the actual rendering of graphics objects on the screen.

→ The GPU processes the commands received from the application & performs the necessary calculations & rasterization to generate the final images.

→ The GPU is optimized for parallel processing, which allows it to handle large amounts of graphical data efficiently & produce real-time graphics.

OpenGL API:

→ It consists of a set of functions & commands that the application uses to interact with the GPU. These functions are grouped into several categories:

1- Initialization & Configuration

→ functions for initializing OpenGL & setting up the rendering context, including selecting the display mode, creating a rendering window, & specifying attributes like color depth & double buffering.

2. Drawing primitives
→ OpenGL provides functions to draw basic geometric shapes, known as primitives.
→ These include points, lines & polygons (triangles, quads, etc).

3. Transformation & Viewing
→ Functions for applying transformations to objects, such as translation, rotation, scaling and projection.
→ These transformations define the position & orientation of objects in the 3D scene.

4. Shading & lighting
→ OpenGL supports various shading models, including flat shading & smooth shading, to determine how light interacts with surfaces.
→ It also provides functions to specify lighting properties, such as light source position, intensity, & material properties of objects.

5. Texture mapping
→ OpenGL supports texture mapping, a technique used to apply images or patterns to the surfaces of 3D objects, enhancing their visual realism.

6. Buffers and Buffers objects
→ Functions for creating & managing different types of buffers, such as vertex buffers, color buffers & depth buffers, used to store graphics data efficiently.

7. Rendering & display
→ functions for managing the rendering pipeline & handling the process of rendering graphics objects onto the screen.

Q) Explain the GLU, GLUT & GLU categories of APIs used in OpenGL for rendering Graphical objects.

→ In openGL, GLU (openGL Utility Library) and GLUT (openGL Utility Toolkit) are additional libraries that provide higher-level functionalities & utility functions to simplify certain aspects of the rendering process & handle user interaction.

→ These libraries are not part of the core OpenGL API but are commonly used alongside it to make graphics programming more convenient & platform-independent. Here's a brief explanation of GLU, GLUT & GLU categories of APIs:

1. GLU (OpenGL Utility Library):

→ GLU is a library that provides utility functions & higher-level operations built on top of the core OpenGL API.

→ It extends the capabilities of OpenGL by offering functions for handling complex geometric transformations, setting up projection matrices, & tessellating complex shapes.

→ GLU also includes functions for performing coordinate transformations, perspective projections & calculating normals for smooth shading.

→ One of the most significant contributions of GLU is its support for NURBS (Non-Uniform Rational B-splines).

2. GLUT (OpenGL Utility Toolkit):

→ GLUT is another utility library that provides a platform-independent API for handling window creation, user input, & event handling in OpenGL applications.

→ It abstracts the process of window management allowing developers to create windows & handle basic user interactions without about platform-specific details.

→ It simplifies the process of setting up an OpenGL context, managing display refresh handling keyboard & mouse events, & managing window resizing.

→ Due to its simplicity & ease of use, GLUT is often used in educational settings & as a starting point for learning OpenGL.

3. GLUT & GLU categories of APIs:

→ GLUT & GLU are distinct libraries with their own sets of functions, but they are often mentioned together because they complement each other & are commonly used in combⁿ with the core OpenGL API.

→ While OpenGL focuses on the core graphics rendering, GLU & GLUT handle various utility tasks that are common in graphics applications such as window mgmt, user input, & complex transformations.

Q6 What are different types of data structures that can be used for representing graphical objects?

→ Various data structures can be used for representing graphical objects, depending on the complexity & requirements of the graphics application.

→ Here are some common data structures used for this purpose:

1. Points

→ The simplest graphical object representation is a single point in space, represented by its (x, y, z) coordinates in 2D or 3D space.

2. Lines & line segments:

→ Lines can be represented using two points or as a parametric equation of the line. Line segments are a subset of lines & are commonly represented by their two endpoints.

3. Polygons:

→ A polygon is a closed shape consisting of multiple line segments, where the last point is connected to the 1st point.

→ Common representations include:

1) Vertex list
2) Edge list
3) Boundary Representation (B-Rep)

4. Curves

→ Curves can be represented in various forms, such as:

1) Bezier curve

→ Defined by control points that influence the curve's shape.

2) B-splines

→ Piecewise-defined curves defined by the blending functions & control points.

5. Meshes

→ Meshes are used to ~~represent~~ 3D objects and are composed of triangles, quadrilaterals, or other polygons.

6. Octrees & QuadTrees

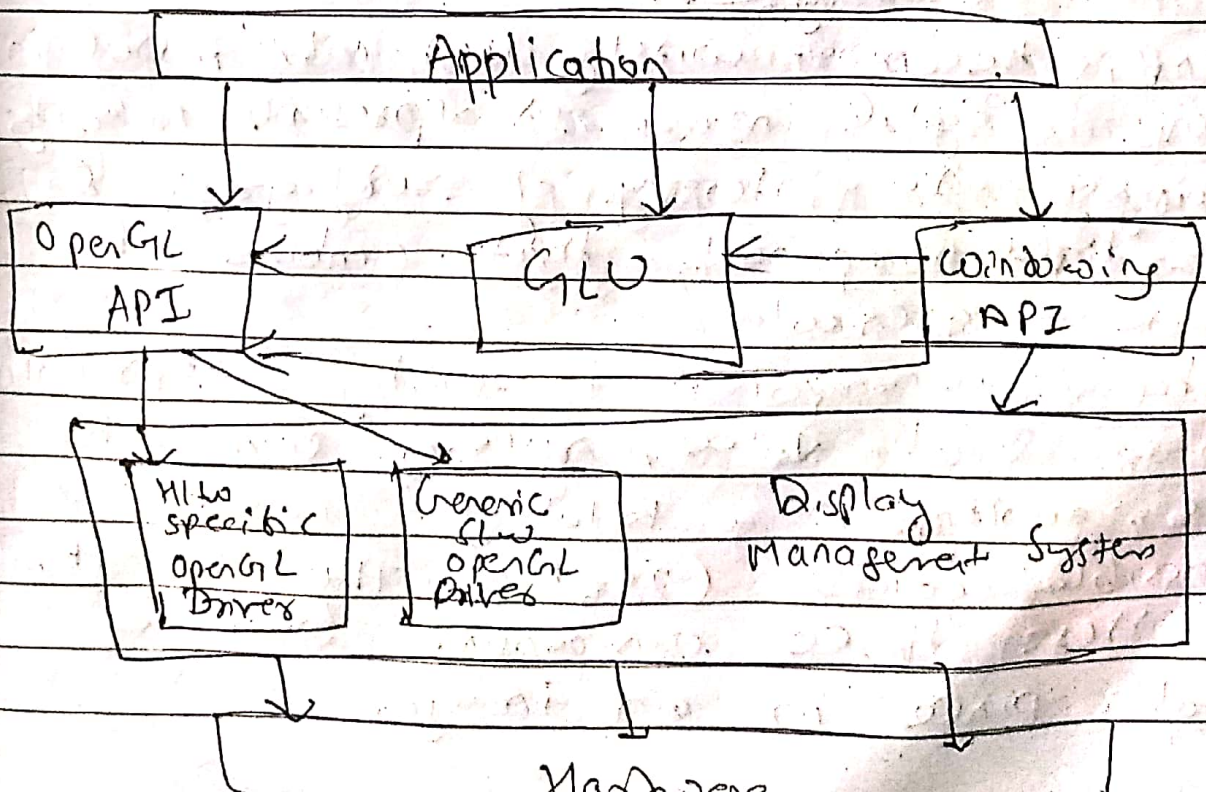
7. Voxel Grids

8. Scene Graphs

9 Parametric surfaces

(Q7) Explain diff types of color models.

Architecture of openGL

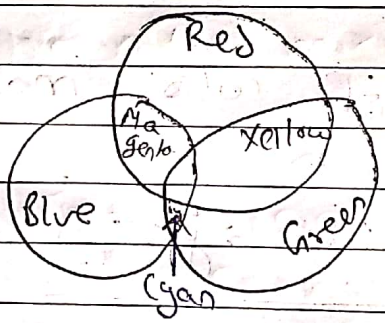


→ In CG, color models are mathematical representations used to define & specify colors in a digital image.

→ These color models provide a standardized way to represent & manipulate colors allowing computers to display, store & process images accurately.

Here are some common types of color models used in CG:

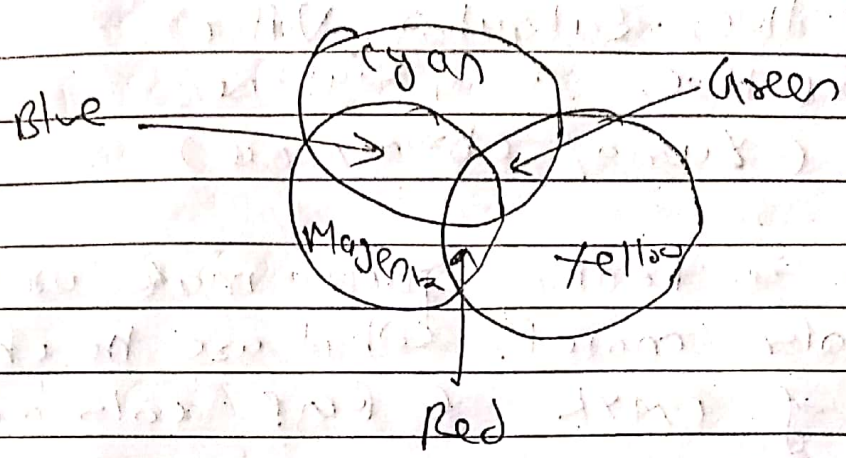
2) RGB (Red, Green, Blue)



→ RGB is the most widely used color model in CGs & digital images. It represents colors by specifying the intensity of red, green, & blue light components that combine to create a given color.

→ Each color component is usually represented as an 8-bit value, ranging from 0 to 255, resulting in a total of 16.8 million possible colors (256^3). The combination of these three components forms a color pixel in an image.

2) CMYK OR "PROCESS COLOR"



C = Cyan
M = Magenta
Y = Yellow
K = Key (Black

→ CMYK color model is a subtractive color model used in printing. It represents colors by specifying the amount of cyan, magenta, & yellow inks needed to subtract from white to produce a specific color.

→ CMYK is an extension of the CMY model & adds a black (key) component. It is commonly used in color printing to achieve a more accurate representation of colors by using black ink to improve contrast & save on the cost of colored inks.

Others:

3. HSV (Hue, Saturation, Value)
4. HSL (Hue, " , Lightness)
5. YUV (Luma, Chrominance)

(Q) Why does a Printing press make use of CMYK color model? What are the characteristics of CMYK & RGB color models?

→ A printing press makes use of the CMYK color model because

- It is a subtractive color model that closely matches the color mixing process used in printing.
- In subtracting color mixing, colors are created by subtracting different wavelengths of light from white.
- When colored inks are applied to paper, they absorb or subtract certain colors resulting in the perception of a specific color to the human eye.

Preferred?

1) Subtractive color mixing
2) Printing Efficiency
3) Color Reproduction.

Q5) What is a callback function in OpenGL? How are they used?

→ In openGL, a callback function is a user-defined function that is registered with the openGL library to be executed in response to specific events or conditions.

→ Callback functions allow developers to handle events, such as user input, window resizing or rendering updates, in a customized manner.

→ These functions are called by the openGL library when the corresponding event occurs, providing a way for developers to extend the functionality of their openGL applications.

Here's how callback functions are used in openGL:

1. Registering callback functions
2. Defining the callback function
3. Handling Events
4. Updating Graphics State.

Common Callback functions in opengl:

1) Display Callback
2) Keyboard "
3) Mouse "
4) Reshape "
5) Idle "

Eg:

```
#include <windows.h>
#include <gl/glut.h>
#include <gl/glu.h>

void display(){
}

void main(int argc, char **argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|
                        GLUT_RGB);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(100, 100);
    glutCreateWindow("suchak");
    glutDisplayFunc(display);
    glutMainLoop();
}
```

// CLEARING THE BACKGROUND

```
void display (){
    glClearColor ( 1.0, 0.0, 1.0, 1.0);
    glClear ( GL_COLOR_BUFFER_BIT);
    glFlush ();
}
```

// PLOTTING A POLYGON: ADDED INSIDE DISPLAY # FUNCTION

```
glShadeModel (GL_SMOOTH);
  glBegin (GL_POLYGON);
    glVertex3f ( 0.25, 0.25 , 0.0);
    glVertex3f ( 0.75, 0.75, 0.0 );
    glVertex3f ( 0.75, 0.25 , 0.0 );
    glVertex3f ( 0.25, 0.75, 0.0);
  glEnd ();
```

// ADDED TO DRAW A POLYGON

```
void display() {
    glClearColor ( 1.0, 0.0, 1.0, 1.0);
    glClear ( GL_COLOR_BUFFER_BIT);
    glShadeModel (GL_FLAT);
    glColor3f ( 0.0, 1.0, 0.0);
```

Q6) What is diff. bet PHIGS, GKS & OpenGL?

→ PHIGS, GKS & OpenGL are all graphics libraries or standards used for CG rendering, but they differ in their approach, functionality, & historical context.

a) PHIGS

→ Programmer's Hierarchical Interactive Graphics System

→ PHIGS is an older standard for interactive 2D & 3D CGs, primarily developed in the 1980s.

→ It was designed to be a comprehensive & extensible graphics system, supporting hierarchical modeling, sophisticated transformations, rendering attributes, & event handling.

→ PHIGS focused on providing a standard for graphics systems used in various applications, including computer-aided design (CAD), simulators, & scientific visualization.

→ Its hierarchical structure allows for easy manipulation of complex graphical scenes, & it supports advanced features like clipping, lighting & anti-aliasing.

→ PHIGS faced challenges with performance & hw compatibility, leading to the emergence of more efficient & platform-specific graphics libraries like openGL.

2. GKS

→ Graphical kernel system

→ GKS is another older standard for CGI, developed in the late 1970s & early 1980s.

→ It aimed to provide a universal, device-independent graphics standard, allowing applications to be written once & run on various graphics devices without modification.

→ It supports basic 2D graphics primitives like
- points, lines & polygons

& is intended for simple graphics applications, rather than high-end rendering or 3D graphics.

→ GKS was widely used in scientific & engineering fields for creating graphs & chart.

3. OpenGL

↳ Open Graphics Library

→ It is a modern, platform-independent graphics library developed in early 1990s & has become one of the most widely used graphics APIs.