Normally, the User or Object Coordinate System.

**World Coordinate Window**: A subset of the world coordinate system, defined by the application.

For instance, this could be the coordinate system of reals.
Normalized Device Coordinate System (NDC)

- The standardized coordinate system for all devices.
- Ranges from \((0.0 - 1.0)\) in each dimension.
- Convenient for all device drivers.
- A convenient space for performain Pick operations.
Device Coordinate System

- Coordinate system of Device.
- Could be different for each device.
- Must specify *device square* for mapping from NDC.
Example Coordinate Systems

World Coordinate Windows

Normalized Device Coordinates

Device Coordinates (example)
A Simple Graphics Package

Graphics Primitives

- Point
- Line, Polyline, Polygon, Rectangle
- Circle, Ellipse
- Marker, Polymarker
- Text
Graphics Primitives

Polyline

PolyMarker

Start Angle

End Angle

Ellipse
Graphics Attributes

- Lines: style (continuous, dashed, dotted), width
- Markers: style, size
- Color (color index or true color - RGB)
- Filled Primitives: style (opaque, transparent), pattern (bitmap/pixmap)
- Text: style (font), size, orientation
- Foreground/Background colors
- Inquiry
Text

Attributes

- Font (Times, Helvetica, Roman)
- Appearance (**Bold**, *italic*, *underlined*)
- Size (points), Orientation
- Inter-character, Inter-line spacing

Implementation

- As Pixmaps
- As geometric descriptions (splines, for instance)
Interaction

Human Factors

- Simple, consistent interaction sequences.
- Not too many options, features.
- Appropriate feedback.
- Error recovery.
Input Interaction: Sampling vs. Event-Driven Processing

- **Polling:** Continuous monitoring of all enabled input devices - CPU intensive, wasteful.

- **Interrupt-driven:** Application interrupted when device state changes - a little tricky.

- **Event-driven:** Application chooses to service the event at its convenience (GKS, PHIGS, X).

![Diagram](https://via.placeholder.com/150)
Event Based Processing

Initialize and generate initial display
do
{
  wait for user selection
  switch (on selection)
  {
    selection 1 : process selection 1
    selection 2 : process selection 1
    ...................................
    ...................................
    selection n : process selection n
  }
} while (!quit)
Transformations/Viewing

- Affine (modeling) transforms (scale, rotate, translate)
- Camera specification
- Projection transformations (parallel (orthographic), perspective)
- Clipping and View volume specification
Lighting

- Local models (Ambient, Diffuse, Specular)
- Light sources (color, position, attenuation), types (spotlights)
- Material colors/properties
- Local vs. Infinite Viewpoint
Raster Operations

- Exploits display hardware features, especially saving and restoring bitmaps.
- BITBLT operations.

Canvases

- Any bitmap or pixmap (2D array of pixels) is a canvas.
- Canvases have size, extent, mask and other attributes.
- Can be created, deleted, changed, copied.
- Offscreen or onscreen.
- Operations affect current canvas.
Write Masks:

- Write masks control bit planes that are to be modified
- Logical operations between source and destination canvases.

Application

- Pull down menus.
- Rubber Banding.
Other Package Features

- Pixels, Bitmaps, Images (BitBlt)
- Blending, Anti-aliasing, Fog
- Texture mapping (2D and 3D) and filtering
- Framebuffer Operations
- Modeling and representation (Polygons, Cubics (Bezier, B-Spline, NURBS)
OpenGL: An Introduction

- A graphics language that is a current industry standard
- Widely deployed across multiple application domains, platforms, vendors
- Outgrowth of SGI GL
- Device independent
- Supports interfaces to modern graphics hardware/systems
Graphics APIs

Generally these APIs support both 2D and 3D graphics:

- GKS (Graphics Kernel System)
- PHIGS (Programmer Hierarchical Interactive Graphics Standard)
- PHIGS+
- GL (Graphics Library) by SGI
- OpenGL
- DirectDraw, Direct3D by Microsoft
What is OpenGL

- A software interface to graphics hardware
- Consists of about 250 different commands (200 core, 50 utility functions)
- Implemented on all major platforms/operating systems
- All windowing tasks, user input/output must be supported by underlying windowing system (X, Direct X, etc)
- Limited modeling support - support for low level primitives only
- GLU - higher level utility library with support for modeling features like NURBS.
- OpenGL Utility Toolkit (GLUT) primitive window setup and event handling
GL Output Examples

Wireframe Rendering
Depth Cueing
Anti-aliasing
Flat Shaded Rendering
Smooth (Gouraud) Shaded Rendering
Texture Mapping and Shadows
Atmospheric (Fog) Effects
Rendering Different Material Types
Environment Mapping
What does GL do?

- Construct shapes from geometric primitives (points, lines, polygons, pixels, images)
- Arrange objects in 3D space and select location to view the composed scene.
- Calculate colors of objects, from specification or lighting conditions, pasting texture onto objects.
- Convert mathematical description of objects and colors into screen pixels, termed rasterization.
OpenGL Basic Syntax (C Language Binding)

- **Functions:** `glXXXX` `glBegin`, `glEnd`, `glPolygonMode`
- **Constants:** `GL_XXXX` `GL_2D`, `GL_RGB`, `GL_CCW`
- **Types:** `GLxxx` `GLbyte`, `GLshort`, `GLint`, `GLfloat`, `GLdouble`, `GLboolean`
OpenGL Geometric Output Primitives

- **Types:** GL_POINTS, GL_LINES, GL_LINE_LOOP, GL_LINE_STRIP, GL_POLYGON, GL_QUADS, GL_QUAD_STRIP, GL_TRIANGLE, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN

- **Basic Syntax:**

  ```
  glBegin (PrimitiveConstant);
  glVertex3f (x1,y1,z1)
  glVertex3f (x2,y2,z2)
  .......
  glEnd();
  ```
OpenGL operates as a state machine - state remains in effect until it is changed.

Example states: current color, viewing/projection transformations, line/polygon patterns, light position/characteristics.

Each state variable has a default value and can be queried, \( \text{glGetIntv()} \), \( \text{glGetFloatv()} \), etc. enabled/disabled \( \text{glEnable()} \), \( \text{glDisable()} \).

Can also save collections of attributes \( \text{glPushAttrib()} \), \( \text{glPopAttrib()} \).
OpenGL Rendering Pipeline
Display Lists

- OpenGL has 2 modes of operation: immediate, or retained.
- Display lists allow collections of geometry and attributes to be stored for later use; allows optimization
- Display lists are write-only; no other structure within objects (in contrast to OpenInventor, Java3D, VRML)
Evaluators

- A set of functions that allows evaluation of curved surfaces (Bezier, B-Spline, NURBS), for conversion to points, lines, polygons
- A polynomial mapping, producing surface normals, texture coords, colors, surface coordinates from control points.
Per-Vertex Operations

- Typically, vertices are transformed (modeling, projection) by $4 \times 4$ matrices, in preparation for primitive assembly.
- Texture coordinates can be generated and transformed,
- Lighting calculations are also performed with transformed vertex, normal and lighting attributes.
Primitive Assembly

- Involves clipping, perspective division
- Viewport and depth (Z coord) operations are applied
- Results are complete geometric primitives, in preparation for rasterization.
Points and Lines

```c
glBegin (PrimitiveConstant);
glVertex3f (x1,y1,z1)
glVertex3f (x2,y2,z2)
······
glEnd();
```

ITCS 4120/5120

Graphics Packages and OpenGL
Polygons

![Polygons](image.png)

```c
glBegin (PrimitiveConstant);
glVertex3f (x1, y1, z1)
glVertex3f (x2, y2, z2)
```
**Polygons**

```c
void glBegin (PrimitiveConstant);
    glVertex3f (x1,y1,z1)
    glVertex3f (x2,y2,z2)
    glVertex3f (x3,y3,z3)
    .......
void glEnd();
```
Why glBegin/glEnd?

- Trying to define a separate function for each possible combination of vertex data too cumbersome:
- Last specified color, normal, texcoord used by vertex call.
- versions of calls based on type, eg., vertex3f(), vertex2i().

```c
glBegin (PrimitiveConstant);
    glColor3f (c1,c1,c1)
    glTexCoord2f(t1,t2)
    glVertex3f (x1,y1,z1)
    glColor3i (c1,c2,c3)
    glNormal3f (n1,n2,n3)
    glVertex3f (x2,y2,z2)
glEnd()
```
Why `glBegin/glEnd`?

- Allow maximum possible parallelism in host CPU and Graphics Processor

```
    glBegin (PrimitiveConstant);
    compute x1,y1,z1 coordinate
    glVertex3f (x1,y1,z1)

    compute x2,y2,z2 coordinate
    glVertex3f (x2,y2,z2) · · ·
    glEnd()
```

- GPU can compute during CPU calls.
Vertex Arrays

- downside of glBegin/glEnd is lots of function calls
- OpenGL Vertex Arrays
  - drastically reduce function call overhead
  - support most common combinations of vertex data
  - allow sharing of vertex coordinates for adjacent primitives

```c
glEnableClientState (GL_VERTEX_ARRAY);
glVertexPointer ( 3 , GL_FLOAT, 0, vertex_coords);
glDrawElements (GL_QUADS, 24, GL_UNSIGNED_BYTE, vertex_indices );
```
Pixel Operations

- Pixels are unpacked into components, followed by scaling, bias calculations.
- Results are clamped and written into texture memory or sent to rasterization step.
- Pixel data from framebuffer - apply pixel transfer ops (scale, bias, clamping), then packed for storage in system memory.
Pixel Array Primitives

\texttt{glBitmap (width, height, x0, y0, xOffset, yOffset, bit\_array)}

- width, height - size of bit map to draw (in last specified color)
- x0, y0 defines window coordinate of lower-left corner of bit\_array
- xOffset, yOffset added to current raster position (see \texttt{glRasterPos2i}); useful for text output
- bit\_array bits encode in unsigned byte array
- Figure 3.61, Hearn/Baker.
OpenGL Raster Operations

- Raster operations, bitblts (bit-block transfers), pixblts
- glDrawPixels (width, height, dataFormat, dataType, pixel_array)
- glReadPixels (xmin, ymin, width, height, dataFormat, dataType, pixel_array)
- glCopyPixels from glReadBuffer to glDrawBuffer
Textures (images + lots of texture rendering options) can be bound to texture objects; then you can switch between texture objects.

Multiple textures may be assembled into texture objects, for efficient switching.
Rasterization

- Scan conversion of geometry and pixel data into fragments.
- Line, polygon attributes, shading model and coverage calculations are accounted for rastering lines, polygons into pixels.
- Fragment is pixel color + depth value + stencil + accumulation
Fragment Operations

- Series of operations that might result in discarding fragment.
- Texturing, fog calculations, scissor test, alpha test, stencil test, depth buffer (Z) test, are performed (if enabled)
- Followed by blending, dithering, logical ops, masking performed.
- Processed and fragments that survive are drawn into the appropriate buffer, becoming a rendered pixel.