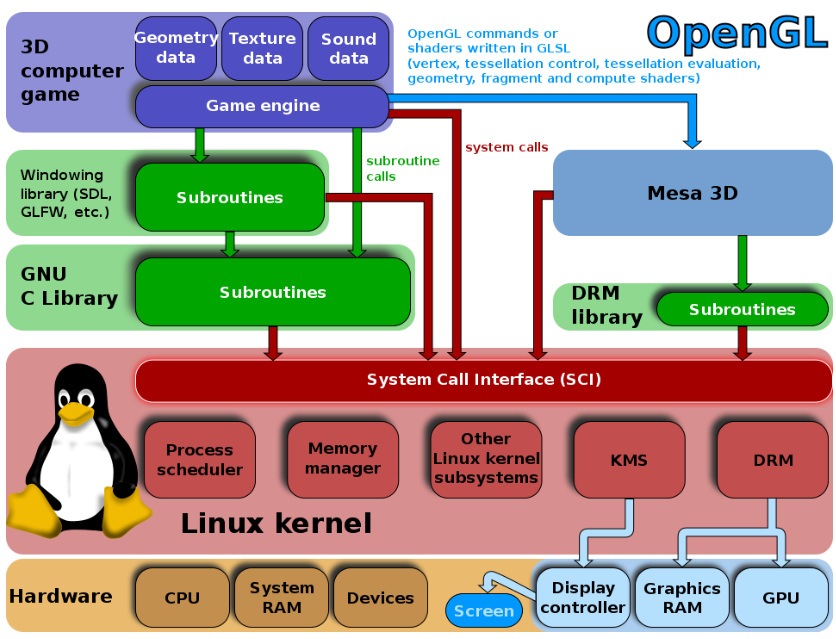
**CPU** (central processing unit) is a generalized processor that is designed to carry out a wide variety of tasks.

**GPU** (graphics processing unit) is a specialized processing unit with enhanced mathematical computation capability, ideal for computer graphics and machine-learning tasks.

While GPUs can process data several orders of magnitude faster than a CPU due to massive parallelism, **GPUs are not as versatile as CPUs**. CPUs have large and broad instruction sets, managing every input and output of a computer, which a GPU cannot do.

**Device Dependent X**: X server will directly communicate to hardware for rendering.

**Device Independent X**: Indirect rendering.



**Processing part: DRM -> Graphics Ram & GPU**

**Display part: KMS-> Display controller -> screen**

**Libdrm:**  provides a user space library for accessing the DRM, direct rendering manager, on operating systems that support the ioctl interface.

**DRM**: The Direct Rendering Manager is a subsystem of the Linux kernel responsible for interfacing with GPUs and Graphics RAM/Card.

**KMS**: Kernel Mode Setting (KMS) is **a method for setting display resolution and depth in the kernel space rather than user space.**

**Mesa**: is **an open-source implementation of OpenGL, Vulkan, and other graphics API specifications**.

**OpenGL:** Opengl provides sets of APIS for rendering 2D and 3D vector graphics.

We can use OpenGL or DirectX but opengl is platform independent.

**Shaders**: Shaders produce lit and shadowed image.

**For example we have six numbers as input and o/p will be image->**

Diagram, timeline

Description automatically generated

**Primitive processing**:

**Primitive assembly:** Identify the required triangles\lines\quad.

**Fragment Shader**: is the Shader stage that will **process a Fragment generated by the Rasterization into a set of colors and a single depth value**

**Rasterization**: is the act of converting an image described in a vector format(shapes) into a raster image (pixels or dots) to display on a video device, print it or to save it in a bitmap file format.

**Vertex Shaders: transform shape positions into 3D drawing coordinates.**

**Vertex shaders describe the attributes (position, texture coordinates, colors, etc.)** **of a vertex, while pixel shaders describe the traits (color, z-depth and alpha value) of a pixel.**

In digital images, each pixel contains color information (such as values describing intensity of red, green, and blue) and also contains a value for its opacity known as its 'alpha' value. **An alpha value of 1 means totally opaque, and an alpha value of 0 means totally transparent**.

**Culling:** the act or process of selecting and removing desirable or undesirable individuals from a group.

In OpenGL, an object is made up of geometric primitives such as triangle, quad, line segment and point. A primitive is **made up of one or more vertices**. OpenGL supports the following primitives: A geometric primitive is defined by specifying its vertices via glVertex function, enclosed within a pair glBegin and glEnd .

OpenGL’s native language is C, so a C++ program can make direct OpenGL

function calls. There are 3 variants (opengl, opengles, webgl).

OpenGL doesn’t draw to a computer screen. Rather, it renders to a

*frame buffer*, and it is the job of the individual machine to then draw the contents

of the frame buffer onto a window on the screen.

GLSL is an example of a *shader language*. Shader languages are intended to run on a GPU, in the

context of a graphics pipeline.

**glClearColor() command specifies the color value to be applied**

**when clearing the background—in this case (1,0,0,1), corresponding to the RGB**

**values of the color red (plus a “1” for the opacity component). We then use the**

**OpenGL call glClear(GL\_COLOR\_BUFFER\_BIT) to actually fill the color buffer with**

**that color. Opacity is transparency.**

* **In the constructor, we call QGLWidget::setFormat() to specify the OpenGL**

**display context, and we initialize the class’s private variables.**

Ex: Tetrahedron::Tetrahedron(QWidget \*parent)

: QGLWidget(parent)

{

setFormat(QGLFormat(QGL::DoubleBuffer | QGL::DepthBuffer));

rotationX = -21.0;

rotationY = -57.0;

rotationZ = 0.0;

faceColors[0] = Qt::red;

faceColors[1] = Qt::green;

faceColors[2] = Qt::blue;

faceColors[3] = Qt::yellow;

}

* **The initializeGL() function is called just once, before paintGL() is called. This**

**is the place where we can set up the OpenGL rendering context, define display**

**lists, and perform other initializations.**

**Ex:** void Tetrahedron::initializeGL()

{

200 *8. 2D and 3D Graphics*

qglClearColor(Qt::black);

glShadeModel(GL\_FLAT);

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_CULL\_FACE);

}

glShadeModel — select flat or smooth shading: void **glShadeModel**(GLenum mode);

Accepted values for GLenum mode are GL\_FLAT and GL\_SMOOTH. The initial value is GL\_SMOOTH.

glEnable — enable or disable server-side GL capabilities: void **glEnable**(GLenum cap);

void **glDisable**(GLenum cap);

* **The resizeGL() function is called before paintGL() is called the first time, but**

**after initializeGL() is called. It is also called whenever the widget is resized.**

**This is the place where we can set up the OpenGL viewport, projection, and**

**any other settings that depend on the widget’s size.**

**Ex:** void Tetrahedron::resizeGL(int width, int height)

{

glViewport(0, 0, width, height);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

GLfloat x = GLfloat(width) / height;

glFrustum(-x, x, -1.0, 1.0, 4.0, 15.0);

glMatrixMode(GL\_MODELVIEW);

}

glViewport — set the viewport:

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

**Description**

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

GL\_MODELVIEW

Applies subsequent matrix operations to the modelview matrix stack.

GL\_PROJECTION

Applies subsequent matrix operations to the projection matrix stack.

GL\_TEXTURE

Applies subsequent matrix operations to the texture matrix stack.

glLoadIdentity — replace the current matrix with the identity matrix(An identity matrix is a **square** matrix having 1s on the main diagonal, and 0s everywhere else)