GRAPHIC RENDERING

JEFWA NGOMBO NICHOLAS

ngombonj@gmail.com

0722641884

GRAPHIC RENDERING

Conversion of the model information into pixels drawn on the screen

Types of rendering

Photorealistic rendering

Texture mapped rendering(non-photorealistic rendering)

Types of RENDERING

- Photorealistic rendering
  - simulating light transport to get physically based realistic photo images
  - Uses ray tracing rendering technique
- Non photorealistic (texture mapped) rendering
  - applying some kind of style
  - Uses rasterization rendering technique

Rendering techniques

- Rasterization
  - Conversion of geometry data or pixel data into fragments

- ray tracing
  - From the image trace a ray of light through a pixel frame to locate where the ray intersects the an object and calculate the color the object at that location for the pixel on the image

RENDERING

Two Approaches

- Start from **geometry**
  - For each polygon / triangle:
    - Is it visible?
    - Where is it?
    - What color is it?

Rasterization

- Start from **pixels**
  - For each pixel in the final image:
    - Which object is visible at this pixel?
    - What color is it?

Ray Tracing

Ray tracing

- The two basic operations are
  - Light transportation
    - –How much light gets from one place to another
  - scattering
    - –How surfaces interact with light.

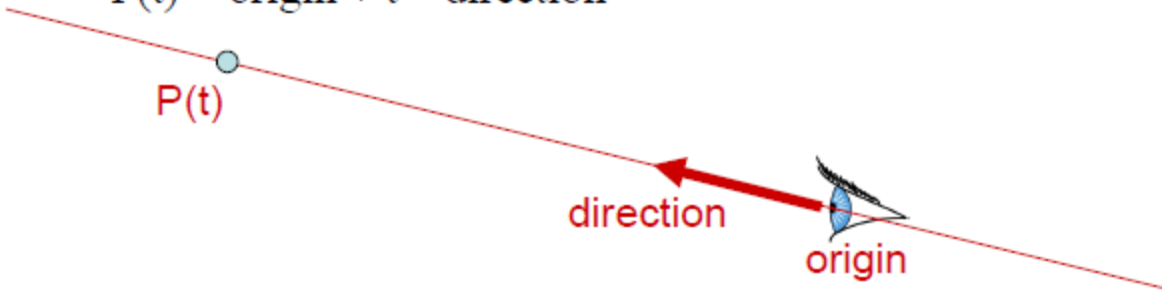
ray tracing

- **ray tracing** is a technique for generating an image by tracing the path of light through pixels in an image plane.
- Ray tracing best is suited for applications where the image can be rendered slowly ahead of time, such as in still images and film and television special effects,
- Ray tracing is more poorly suited for real-time applications like computer games where speed is critical.

Ray representation

- Origin – Point
- Direction – Vector
  - normalized is better
- Parametric line
  - P(t) = origin + t * direction

How would you represent a ray?



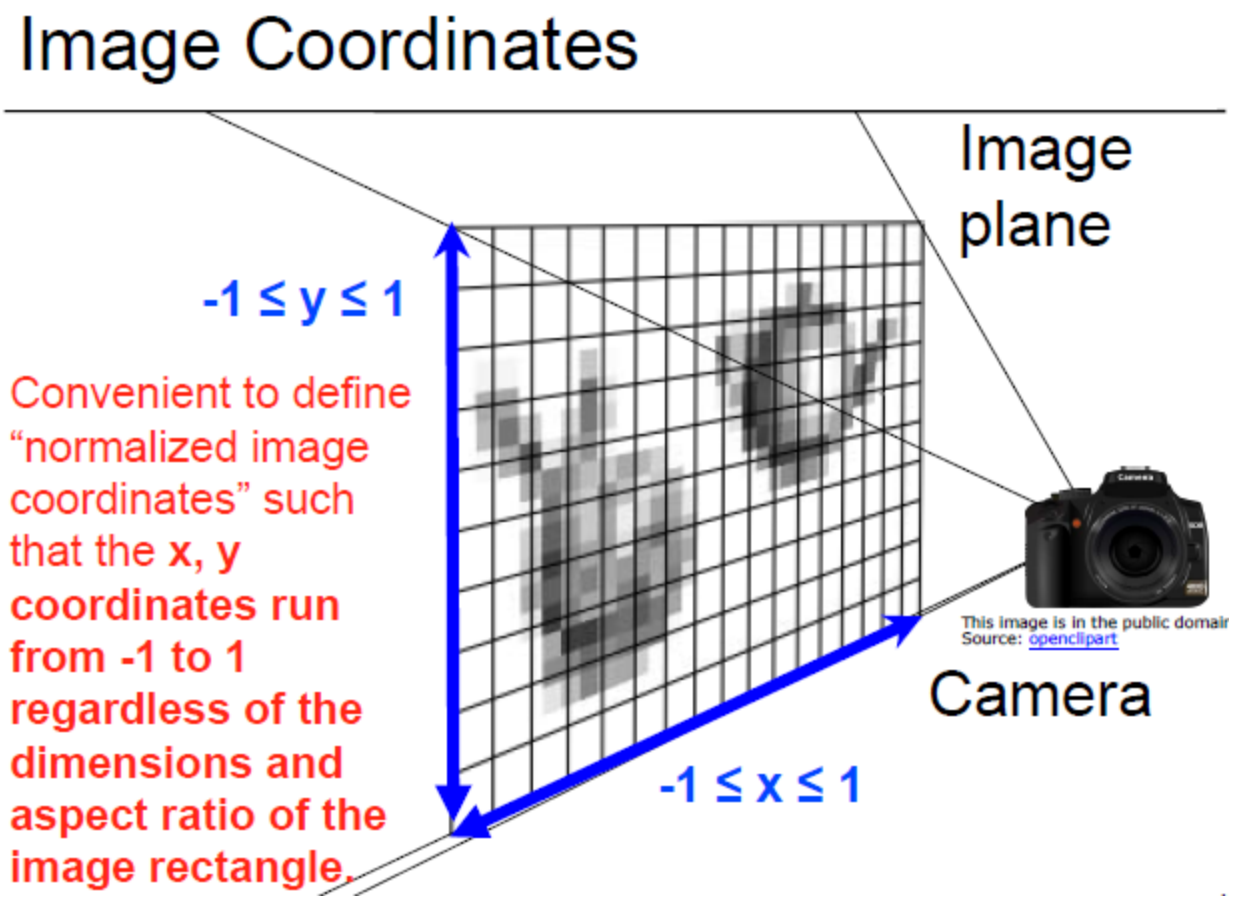
Ray representation

Origin – Point

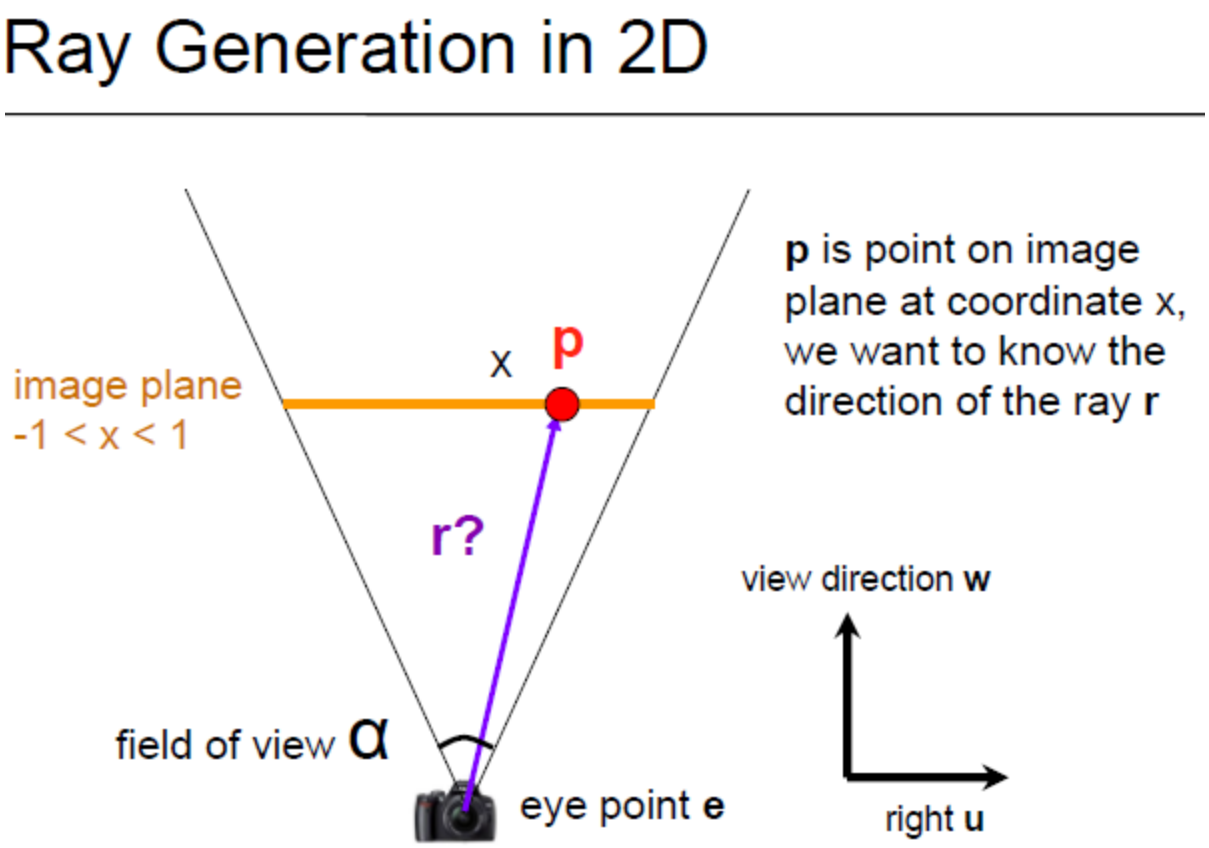
- Direction – Vector
  - normalized is better
- Parametric line
  - P(t) = origin + t * direction

Find smallest t > 0 such that P(t) lies on a surface in the scene

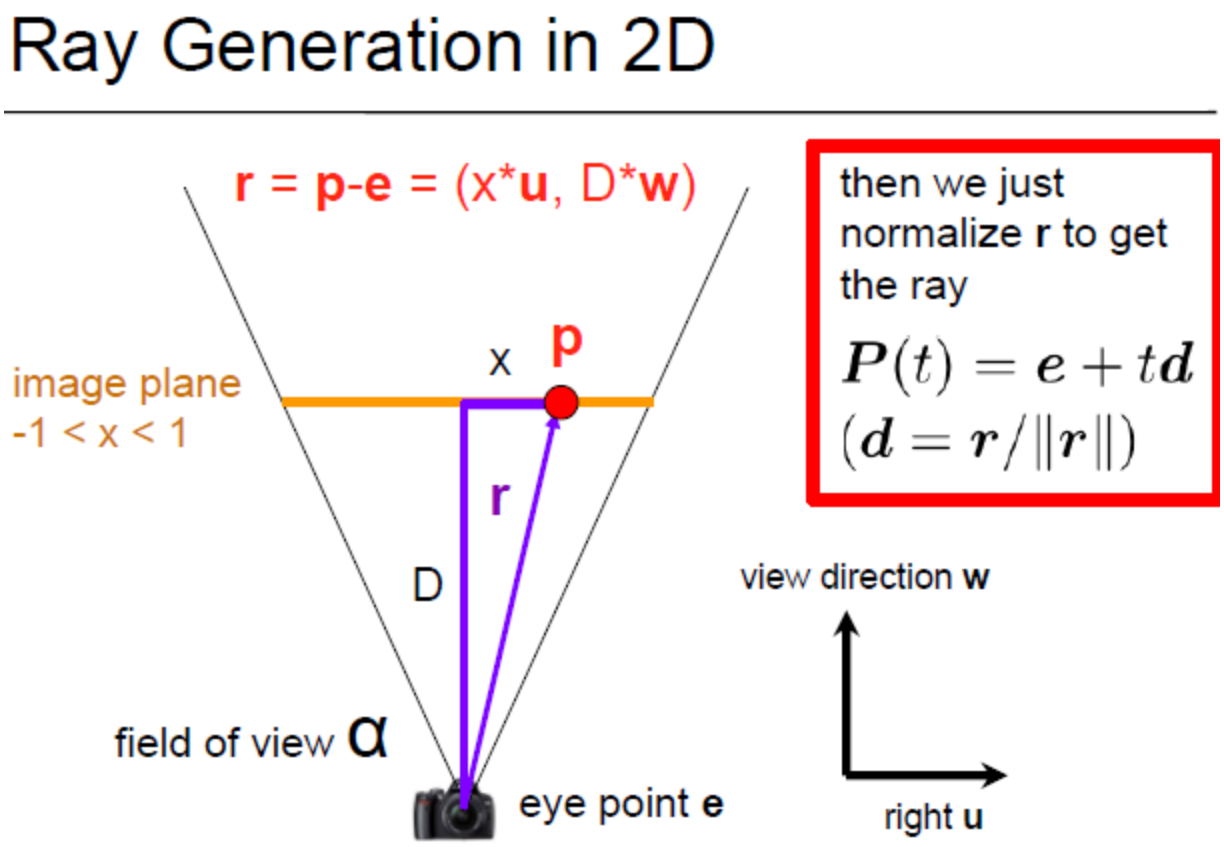
Image coordinates



Ray generation



Ray generation



Rasterization

- -Rasterization is the conversion of both geometric and pixel data into fragments.
- Each fragment square corresponds to a pixel in the framebuffer.

Rasterization pipline

Clip Space Transformation. The first phase of rasterization is to transform the vertices of each triangle into a certain region of space.

Everything within this volume will be rendered to the output image, and everything that falls outside of this region will not be

clip space

The volume that the triangle is transformed into ***clip coordinates***.

The positions of the triangle's vertices in clip space

A position in clip space has *four coordinates* X, Y, Z ,W. The last coordinate defines what the extents of clip space are

for this vertex.

any triangles that fall outside clip space are discarded. Any triangles that

are partially outside clip space undergo a process called *clipping. This breaks the triangle apart into a number of smaller triangles, such that*

the smaller triangles are all entirely within clip space

Window transformation

to transform the vertices of each triangle

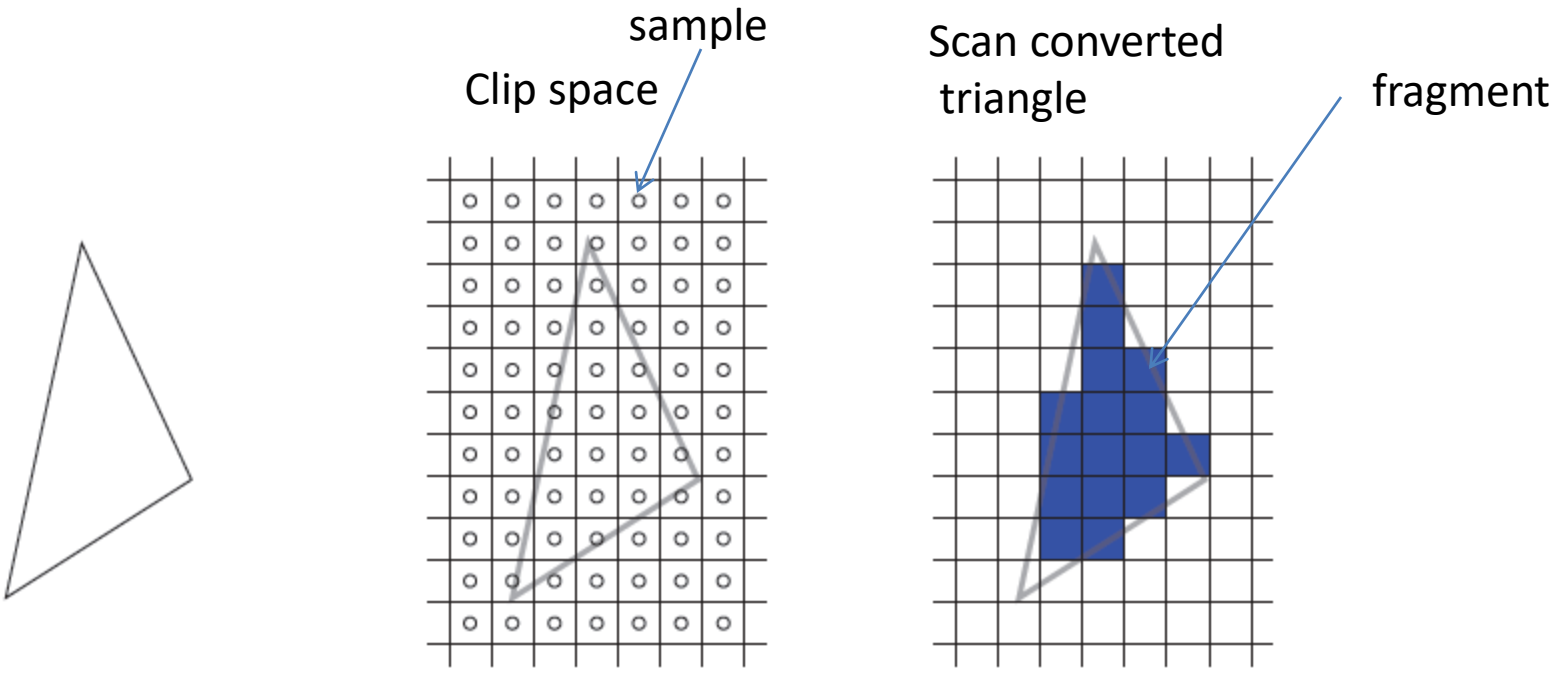
from normalized device coordinates to *window coordinates*

Scan conversion

This process takes the triangle and breaks it up based on the arrangement of window pixels over the output image that the triangle covers

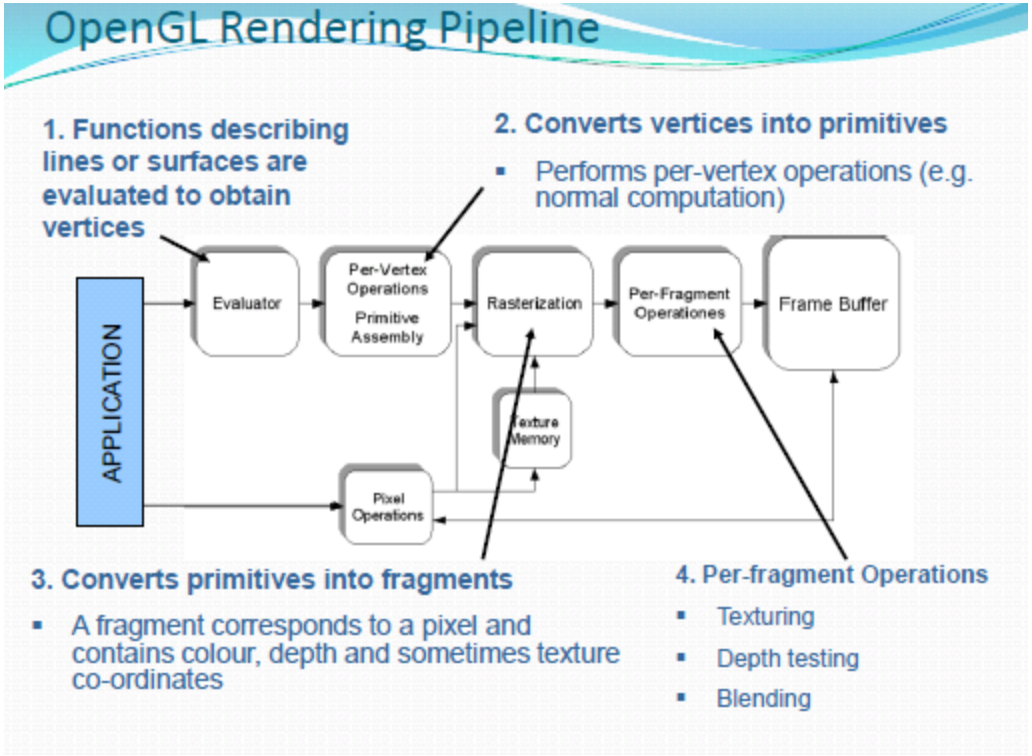
During scan conversion, a triangle will produce a *fragment for every pixel sample that is* within the 2D area of the triangle

Rasterisation illustration

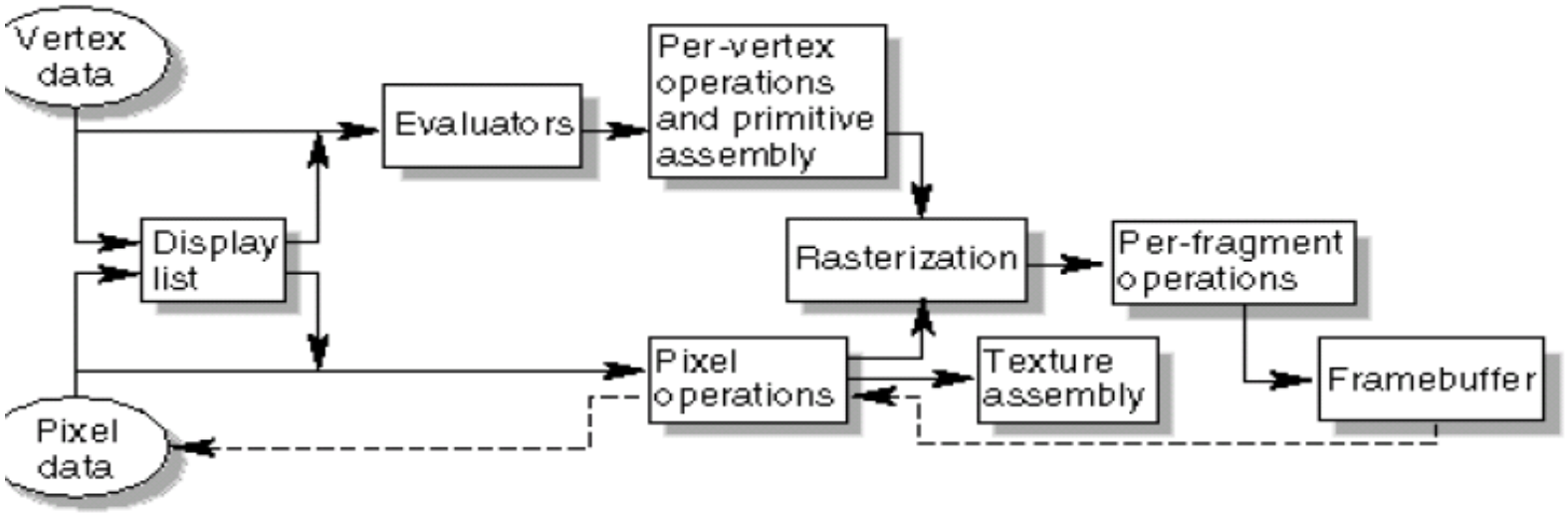


The center image shows the digital grid of output pixels; the circles represent the center of each pixel. The center of each pixel represents a *sample*: a discrete location within the area of a pixel. During scan conversion, a triangle will produce a *fragment* for every pixel sample that is within the 2D area of the triangle.

RENDERING PROCESS



RENDERING PROCESS



Rendering Pipe line

RENDERING THE MODEL

- Rendering process
  - Rendering engine receives results of calculations by Geometry engine
  - The rendering engine creates a wire frame view in which all the vertices that define the polygons are joined by lines or wires
  - The rendering engine determines the objects that are hidden behind other and identifies the pixels to produce for objects, using either z-sorting or z-buffering techniques ,based on camera's view point

Rendering Pipe line- **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.

Rendering Pipe line-**Evaluators**

- -All *geometric primitives* are described by vertices. Although 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

Rendering Pipe line-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 *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.

Rendering Pipe line

- **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.
- 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.

Rendering Pipe line-**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.

Rendering Pipe line-**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

Rendering Pipe line-Fragment Operations

- -Before values are actually stored into the framebuffer, a series of operations are performed that may alter or even throw out fragments.
- 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
- the scissor test,
- the alpha test,
- the stencil test, and
- depth-buffer test
  - the depth buffer is for hidden-surface removal.
- blending,
- dithering,
- logical operation, and
- masking by a bitmask may be performed.