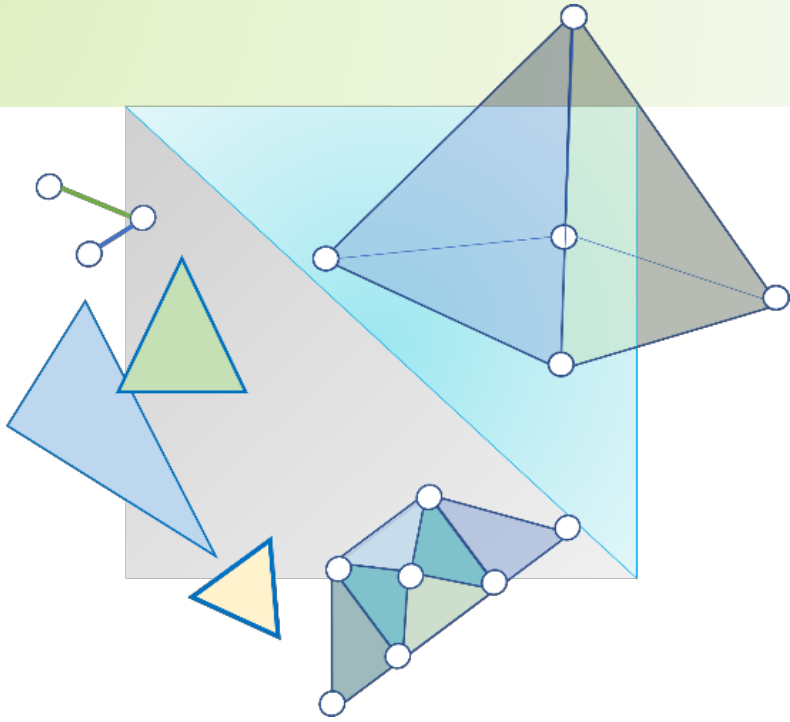
CITS3003 Graphics & Animation



Lecture 3:
Pipeline Architecture

Content

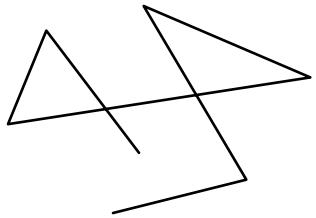
- Expanding on primitives
- Vertex attributes
- OpenGL pipeline architecture
- Immediate mode graphics vs retained mode graphics

OpenGL Primitives

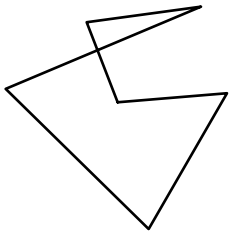
Recall from a previous lecture…

GL_POINTS

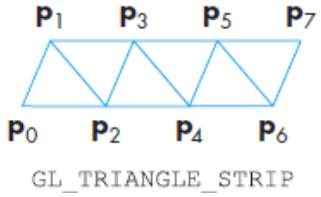
GL_LINES

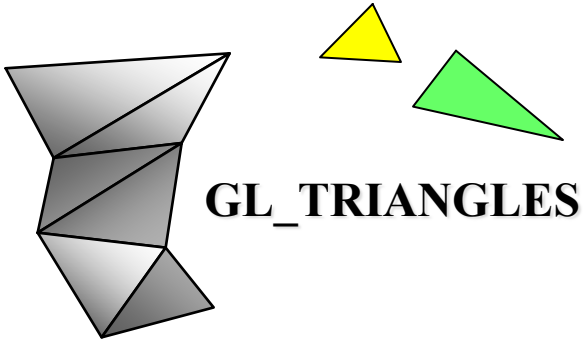


GL_LINE_STRIP



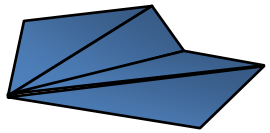
GL_LINE_LOOP



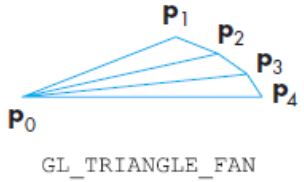


GL_TRIANGLES

GL_TRIANGLE_STRIP



GL_TRIANGLE_FAN



Polygons in OpenGL

Everything you see rendered on the screen is a collection of points, lines, and triangles.

- Applications normally break complex surfaces into a very large number of triangles and send them to OpenGL

The only OpenGL polygons that OpenGL supports are triangles

- Triangles are easy to draw.
- We can approximate surfaces using collections of many triangles.

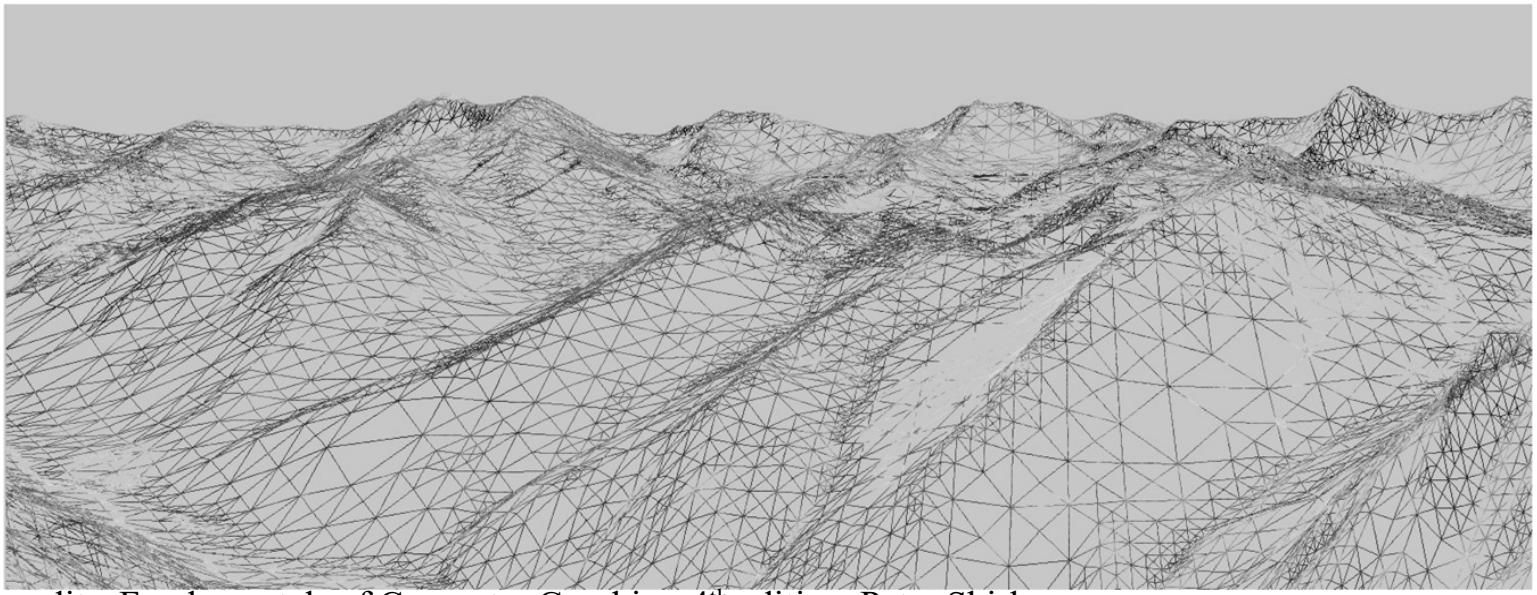
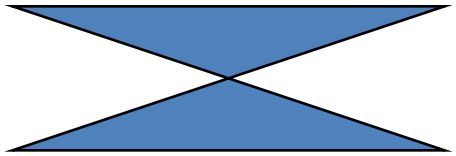


Image credits: Fundamentals of Computer Graphics, 4th edition, Peter Shirley

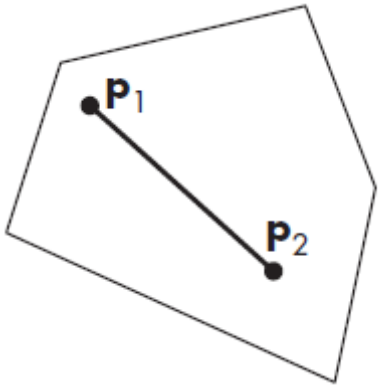
Polygons in OpenGL

- Graphics systems like triangles because triangles are:
  - **Simple**: edges cannot cross
  - **Convex**: All points on a line segment between two points in a polygon are also in that polygon
  - **Flat**: all vertices are in the same plane



Non-simple polygon

nonconvex polygon

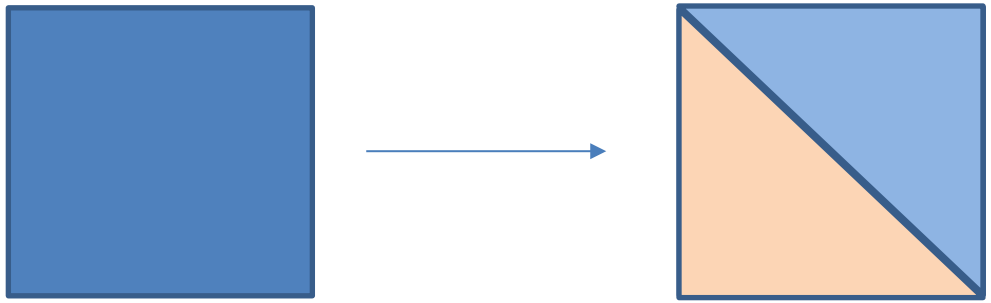


convexity

Polygon Issues (cont.)

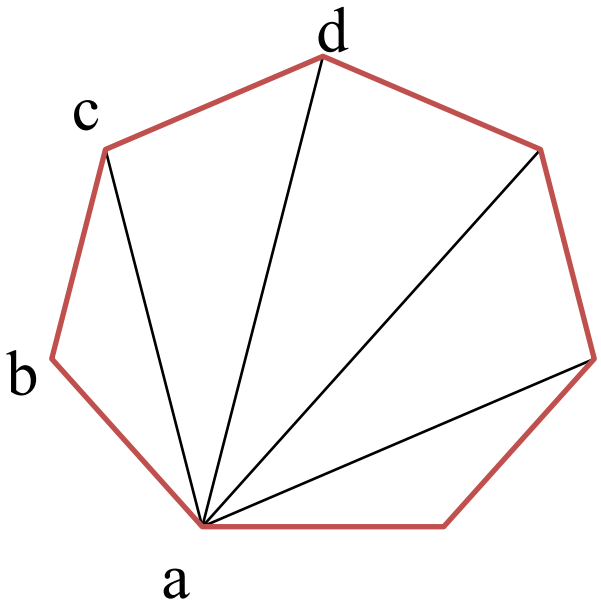
- In practice, we need to deal with more general polygons.
- If other polygons are used, Application program must tessellate a polygon into triangles (a.k.a *triangulation*)
- OpenGL 4.1 contains a tessellator.

Tessellation (tiling) of a flat surface is the process of covering it with one or more geometric shapes (the tiles): Wikipedia



Recursive Triangulation of Convex Polygon

There are some simple algorithms to perform triangulations for planar convex polygons.



- For example:
  1. Start with first three vertices and form a triangle.
     - Start with abc to form the 1st triangle
  2. Remove the second vertex from the list of vertices
     - Remove b
  3. (Recursion) Go to Step 1 to form the 2nd triangle

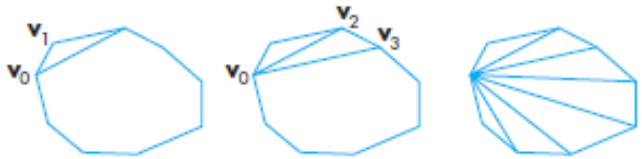


FIGURE 2.18 Recursive triangulation of a convex polygon.

- Does not guarantee all triangles are good.

Triangulation
Good and Bad Triangles

Although every set of vertices can be triangulated, not all triangulations are equivalent.

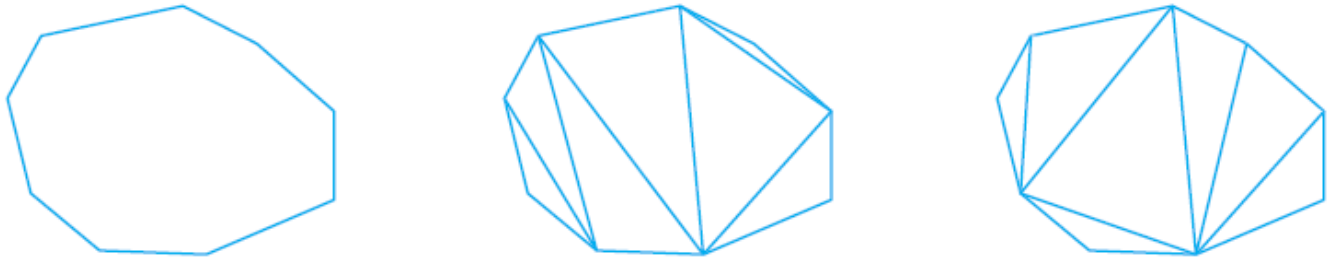
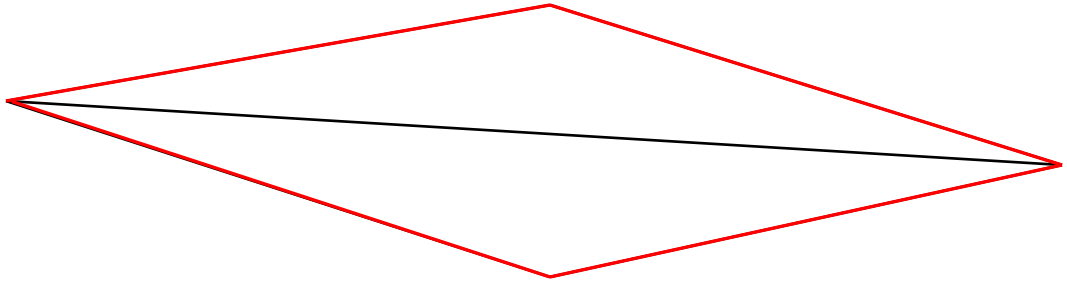


FIGURE 2.16 (a) Two-dimensional polygon. (b) A triangulation. (c) Another triangulation.

Triangulation
Good and Bad Triangles

- Long, thin triangles render badly

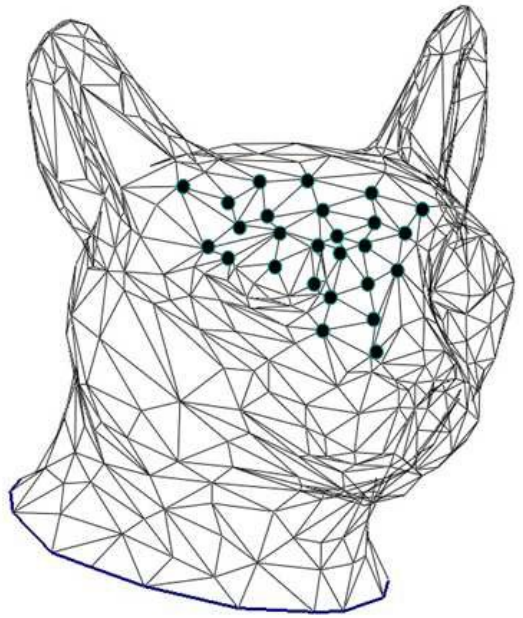


- Equilateral triangles render well
- To get good triangles for rendering
  - ➔ Maximize the minimum interior angle
- **Delaunay triangulation** can be used for unstructured points

Vertices and attributes

- Each triangle is made up of 3 vertices. Each vertex is associated with some numerical data.

- **Vertex Attributes**
  - Each data item associated with a vertex is called an attribute.
    - location of the vertex (using 2 numbers for 2D geometry or 3 numbers for 3D geometry).
    - color (using 3 numbers representing amounts of red, green and blue)
    - material properties etc.



A cat head is described by a soup of triangles. Some of the vertices are highlighted with black dots.

Image credits: SANDER, P., GORTLER, S., SNYDER, J., AND HOPPE, H. Signal-specialized parametrization. In Proceedings of the 13th Eurographicsworkshop on RenderingTechniques (2002), pp. 87–98.. Foundations of 3D Computer Graphics,S.J. Gortler.MIT Press, 2012 , ©Eurographics and Blackwell Publishing Ltd.

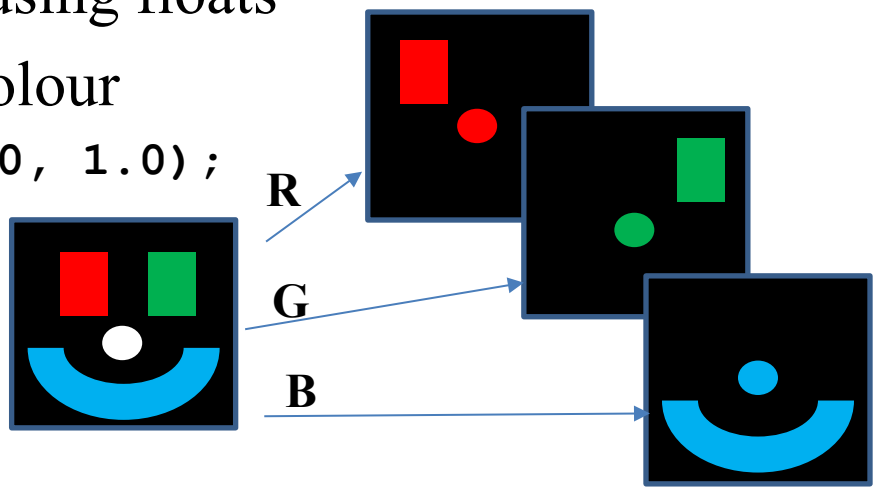
Colour

How is color handled in a graphics system from the programmer's Perspective?

RGB Color model:

- Each colour component is stored separately in the frame buffer.
- Occupies 8 bits per component in the buffer.
- Colour values range:
  - from 0 to 255 using unsigned integers, or
  - from 0.0 (none) to 1.0 (all) using floats
- Use vec3 or vec4 to represent colour

```
vec4 red = vec4(1.0, 0.0, 0.0, 1.0);
```

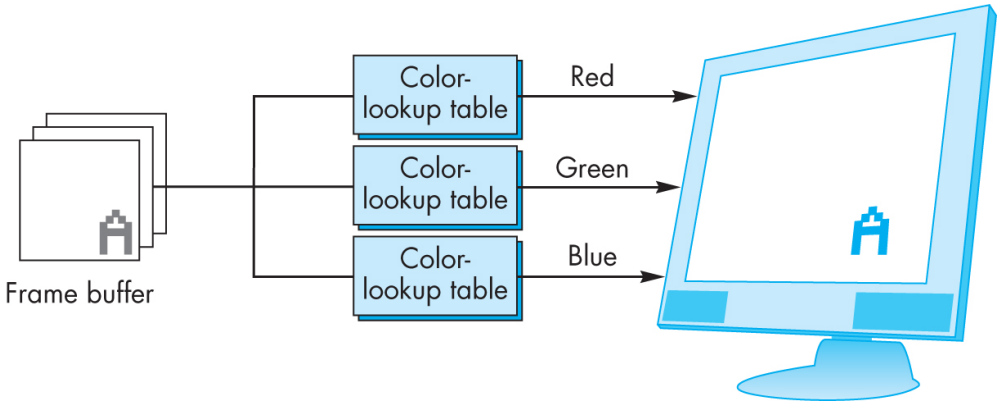


Indexed Colour

- Colours are indices into tables of RGB values
- Requires less memory
  - Indices usually 8 bits
  - no longer part of recent versions of OpenGL

Input	Red	Green	Blue
0	0	0	0
1	$2^m - 1$	0	0
.	0	$2^m - 1$	0
.	.	.	.
.	.	.	.
$2^k - 1$	.	.	.
	m bits	m bits	m bits

FIGURE 2.28 Color-lookup table.

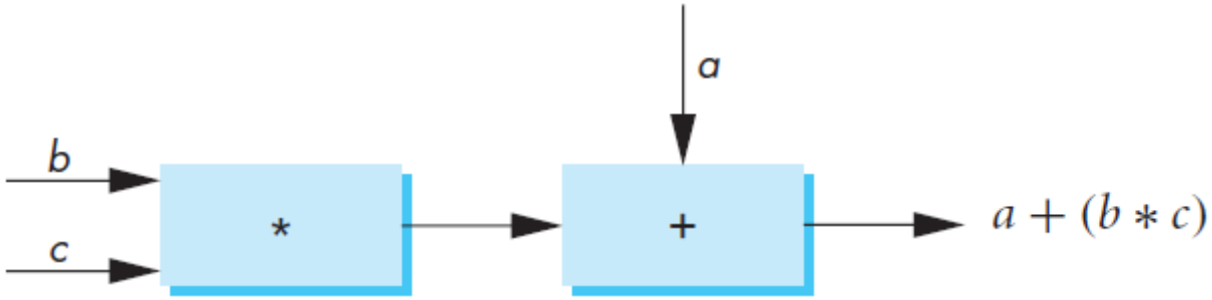


The Graphics Pipeline

Image link

Pipeline Architectures

- Pipeline architectures are very common and can be found in many application domains. E.g., an arithmetic pipeline:

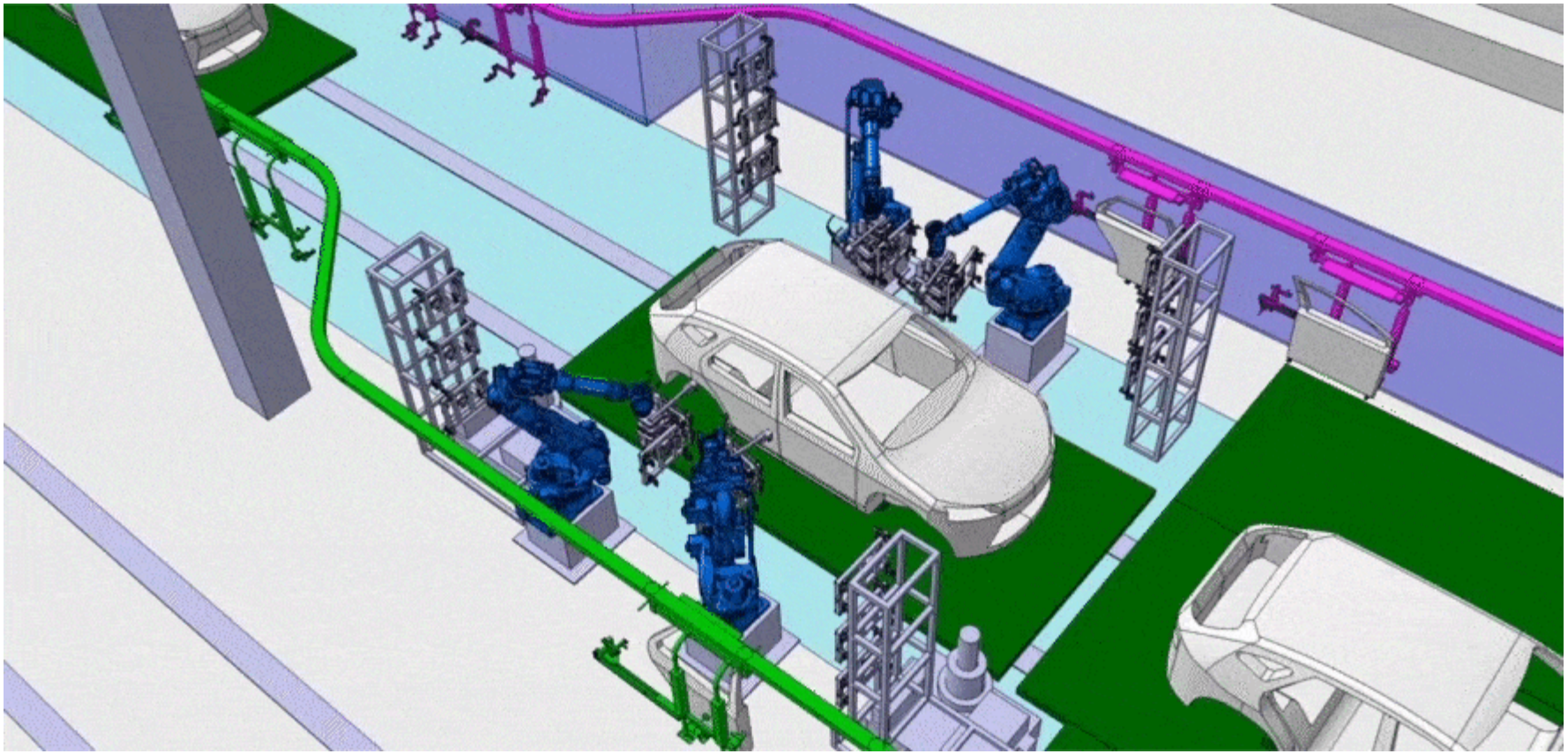


Arithmetic Pipeline

- When two sets of a, b, and c values are passed to the system, the multiplier can carry out the 2nd multiplication without waiting for the adder to finish → the calculation time is shortened!

Pipeline Architectures

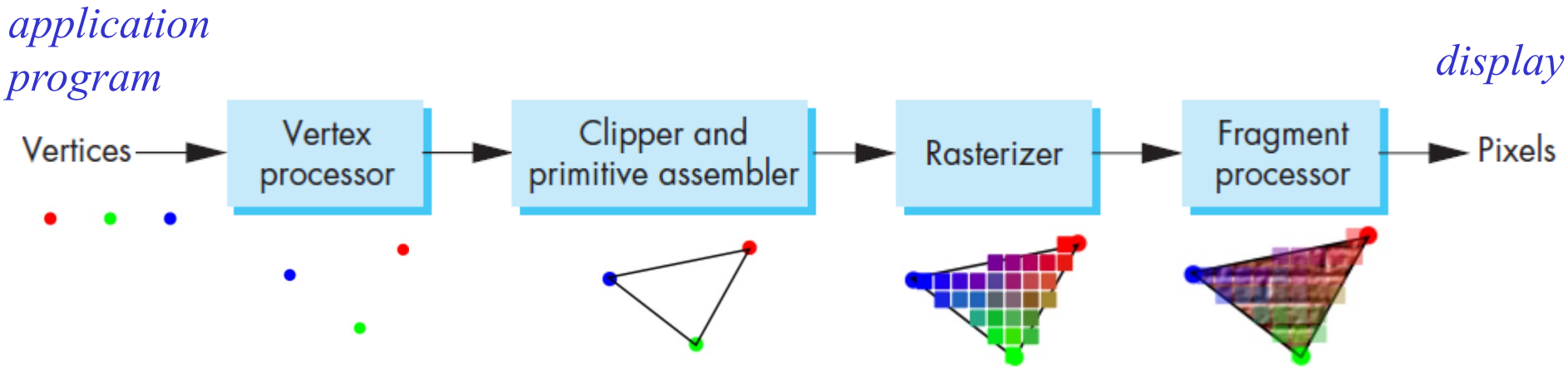
- Pipeline architectures are very common and can be found in many application domains. E.g., an arithmetic pipeline:



Source

OpenGL Graphics Pipeline

- OpenGL utilizes the following pipeline (simplified version):

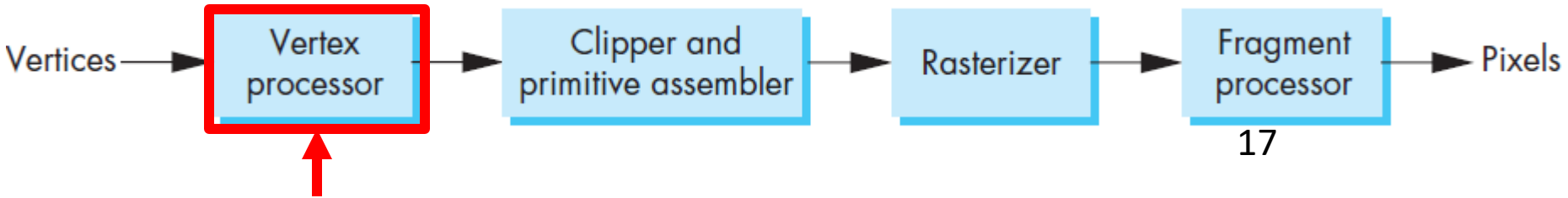


- Objects passed to the pipeline are processed one at a time in the order they are generated by the application program
- All steps can be implemented in hardware on the graphics card

Image credits: Author: Romain Vergne link

Vertex Processing

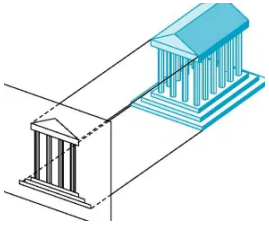
- Much of the work in the pipeline is in converting object representations from one coordinate system to another
  - Object coordinates
  - Camera (eye) coordinates
  - Screen coordinates
- Every change of vertex coordinates is the result of a matrix transformation being applied to the vertices
- Vertex processor can also compute vertex colors

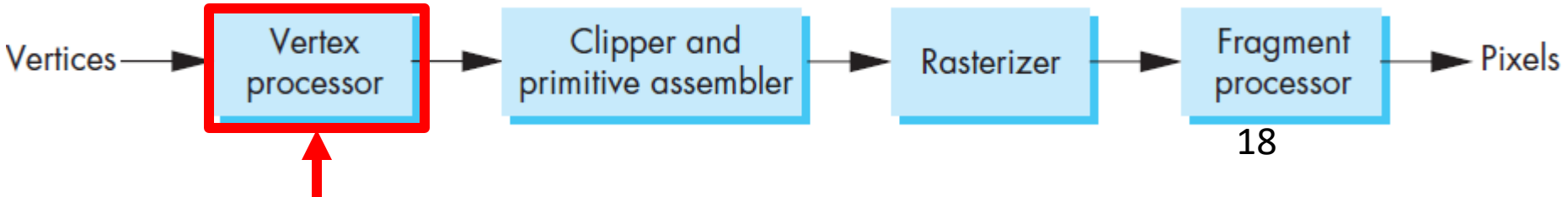


Projection

- *Projection* is the process that combines the 3D viewer with the 3D objects to produce the 2D image

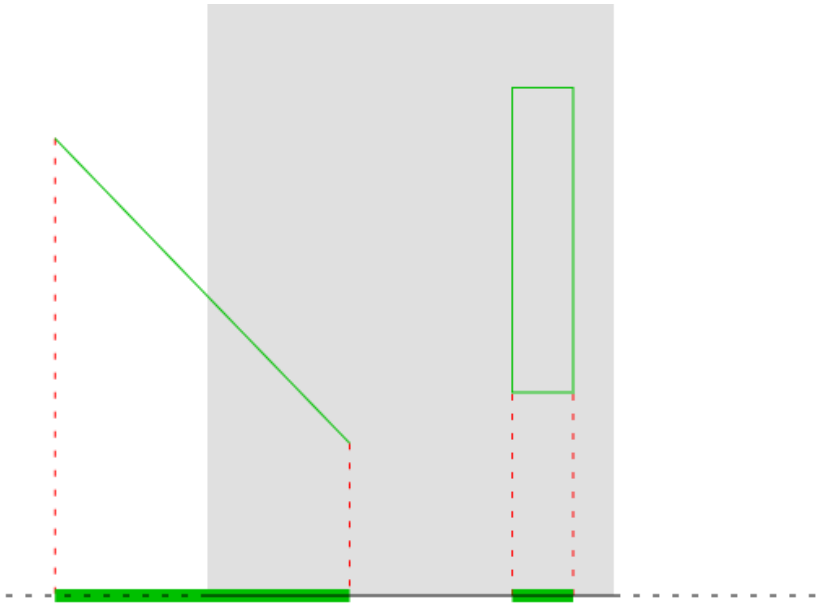
- **Perspective projections:** all projected rays meet at the center of projection
- **Parallel projection:** projected rays are parallel; center of projection is at infinity. (specify the direction of projection instead of the center of projection)



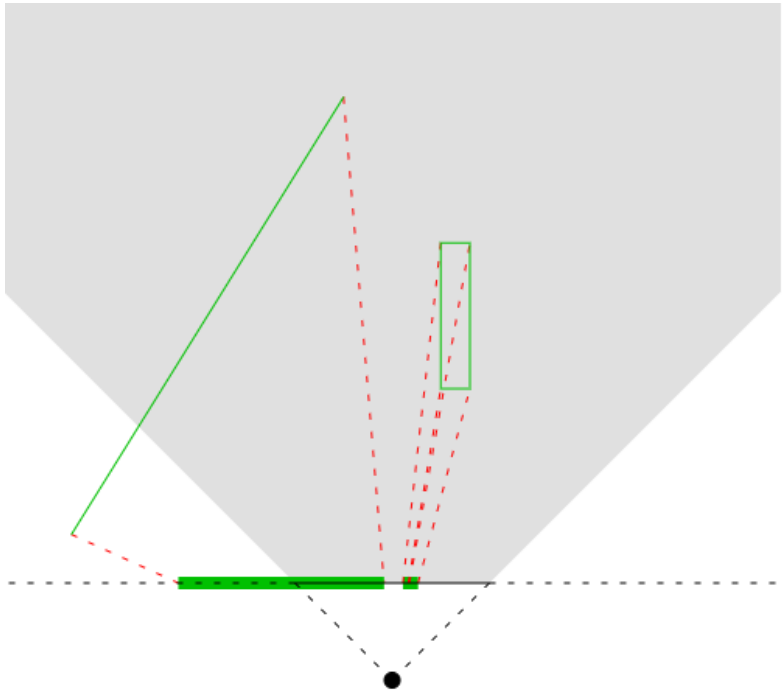


Projection

- Example



2D to 1D Orthographic/Parallel Projection



2D to 1D Perspective Projection

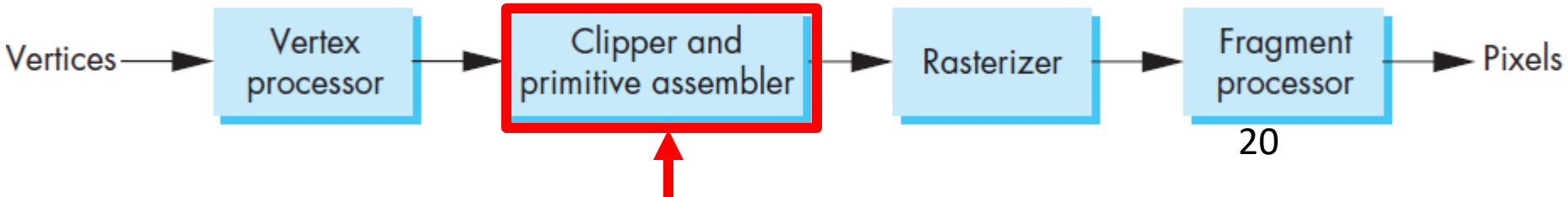
The gray box represents the part of the world that is visible to the projection; parts of the scene outside of this region are not seen

Credits: link

Primitive Assembly

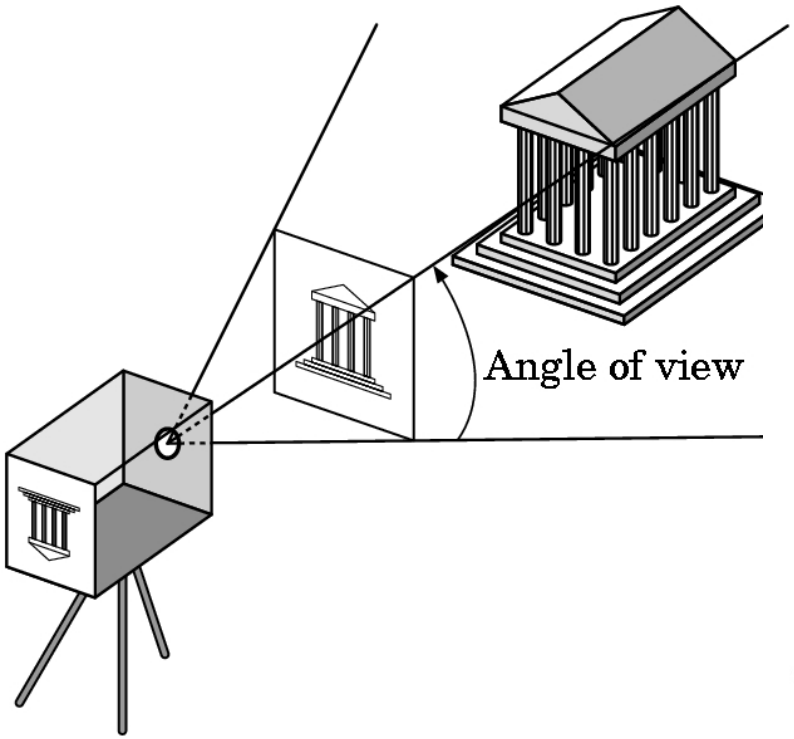
- Vertices must be collected into geometric objects before clipping and rasterization can take place.
  - Line segments
  - Polygons
  - Curves and surfaces

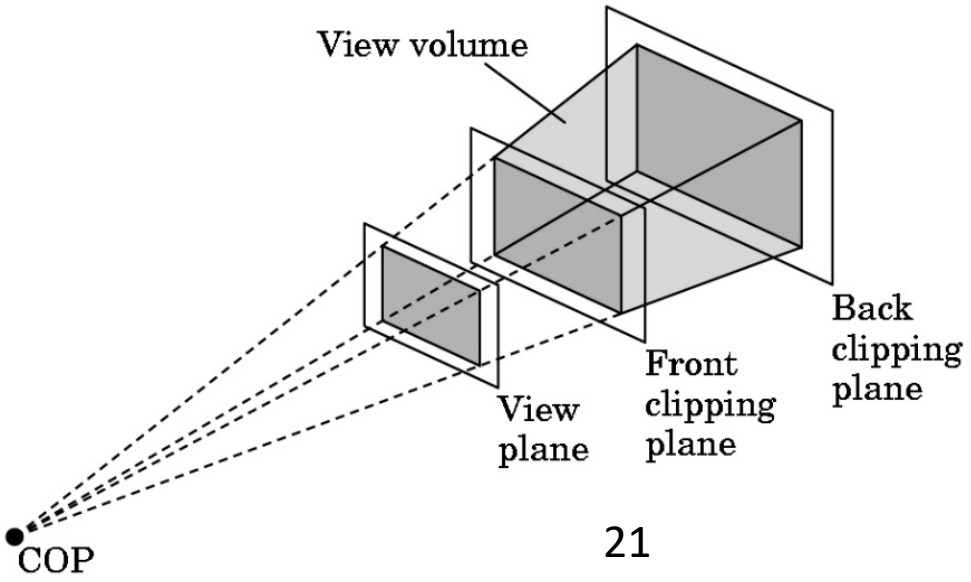
  are formed by the grouping of vertices in this step of the pipeline.



Clipping

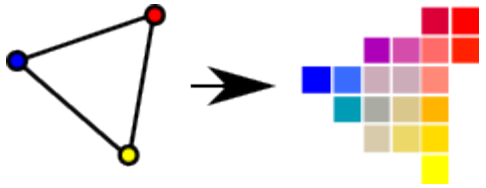
- Just as a real camera cannot "see" the whole world, the virtual camera can only see part of the world or object space
  - Objects that are not within this volume are said to be *clipped* out of the scene





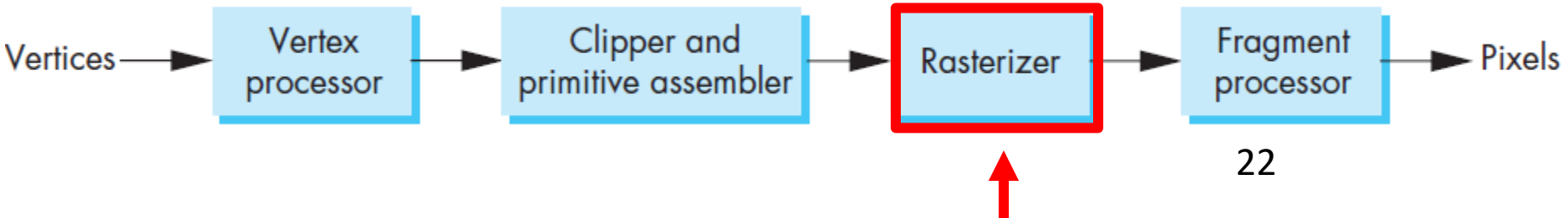
Rasterization

- If an object is not clipped out, the appropriate pixels in the frame buffer must be assigned colors.
- Rasterizer produces a set of fragments for each object.
- Fragments are "**potential pixels**". They:
  - have a location in the frame buffer
  - have colour, depth, and alpha attributes



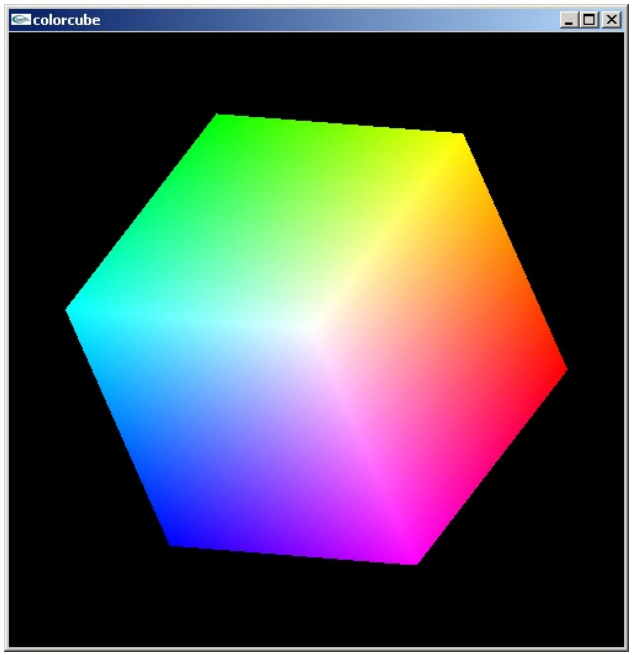
link

- Vertex attributes (colour, transparency) are interpolated over the objects by the rasterizer



Smooth Color

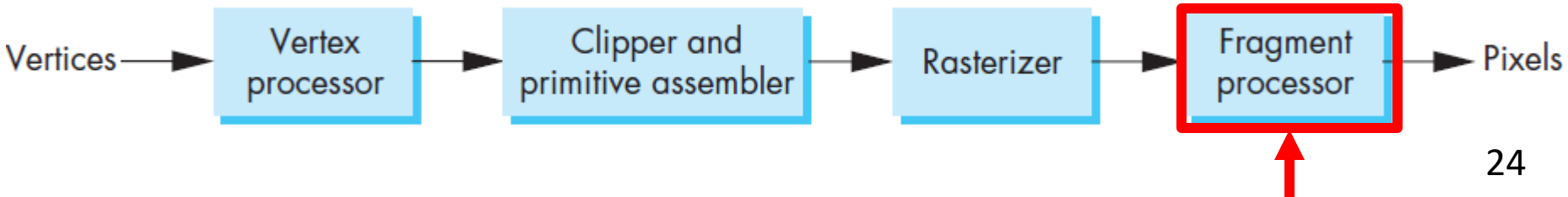
- We can tell the *rasterizer* in the pipeline how to interpolate the vertex colours across the vertices
- Default is ***smooth shading***
  - OpenGL interpolates vertex colors across visible polygon
- Alternative is ***flat shading***
  - Color of the first vertex determines the fill color



- Shading is handled in the fragment shader

Fragment Processing

- Fragments are processed to determine the colour of the corresponding pixel in the frame buffer.
- The colour of a fragment can be determined by texture mapping or by interpolation of vertex colours.
- Fragments may be blocked by other fragments closer to the camera
  - Hidden-surface removal



slido

Choose the correct option(s)

ⓘ Start presenting to display the poll results on this slide.

Retained Mode Graphics

The application constructs a scene from graphics primitives, such as triangles and lines, and the library stores a model of the scene in the memory.

- Put all vertex data in array(s), send array(s) over and store on GPU for multiple renderings

– The application issues commands to update the scene (e.g., add or remove shapes).
– The library is responsible for redrawing the scene.
– A retained-mode API is declarative.

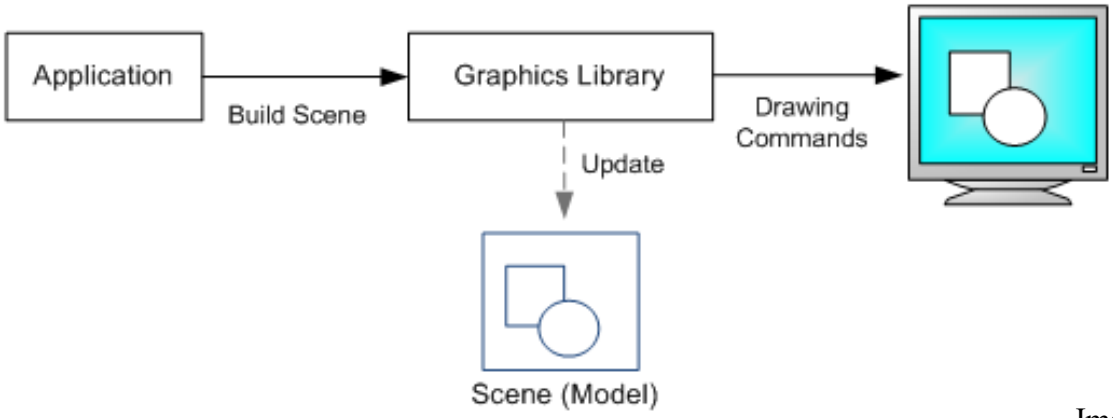
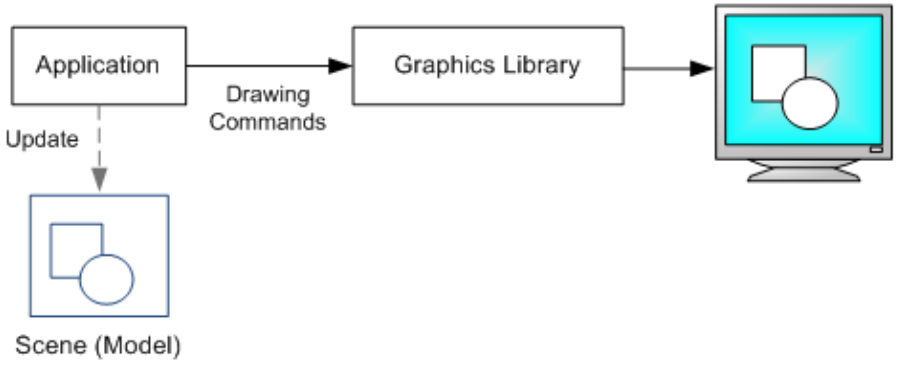


Image taken from docs.microsoft.com

Immediate Mode Graphics

- Older versions of OpenGL adopted **immediate mode graphics**, where:
  - Each time a vertex is specified in application, its location is sent to the GPU.
  - Old style programming, uses **glVertex**
  - The library does not store a scene model between the frames.




```
glBegin(GL_TRIANGLES);
    glVertex3f(-0.5, -0.5, 0.0);
    glVertex3f(0.5, -0.5, 0.0);
    glVertex3f(0.0, 0.5, 0.0);
glEnd();
```

Example code of using immediate mode to draw a simple triangle.

Image taken from docs.microsoft.com

Immediate Mode with OpenGL

- Advantage:
  - No memory is required to store the geometric data (memory efficient)
- Disadvantages:
  - As the vertices are not stored, if they need to be displayed again, the entire vertex creation and the display process must be repeated.
  - Creates bottleneck between CPU and GPU
- Immediate mode graphics has been removed from OpenGL 3.1

Retained Mode Graphics with OpenGL

- Put all vertex and attribute data into an array, send and store that on the GPU
- Update when required
- Retained mode graphics is adopted in OpenGL 3.1 onward.

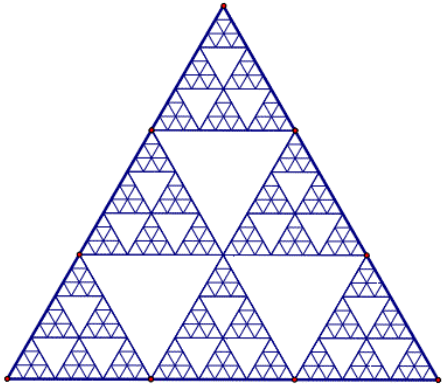
Comparison of the two modes

- Immediate mode graphics

```
main()
{
  initialize_the_system();
  p = find_initial_point();
  for (some_no_of_points) {
    q = generate_a_point(p);
    display(q);
    p = q;
  }
  cleanup();
}
```

- Retained mode graphics

```
main()
{
  initialize_the_system();
  p = find_initial_point();
  for (some_no_of_points) {
    q = generate_a_point(p);
    store_the_point(q);
    p = q;
  }
  display_all_points();
  cleanup();
}
```



2D Sierpinski triangle

Pseudo code for the 2D Sierpinski triangle program for the 2 modes

Further Reading

"Interactive Computer Graphics – A Top-Down Approach with Shader-Based OpenGL" by Edward Angel and Dave Shreiner, 6th Ed, 2012

- Sec. 1.7.2 – 1.7.7 Pipeline Architectures … Fragment Processing
- Sec. 2.1 The Sierpinski Gasket; immediate mode graphics vs retained mode graphics
- Sec 2.4 – 2.4.4 Primitives and Attributes … Triangulation

Graphics Shaders (second edition) Bailey and Cunningham

- Chapter#01