Intro to OpenGL III

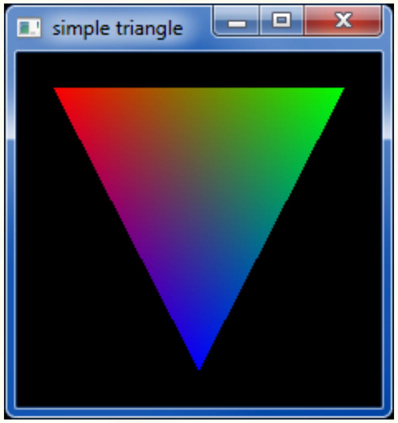
Don Fussell
Computer Science Department
The University of Texas at Austin

Where are we?

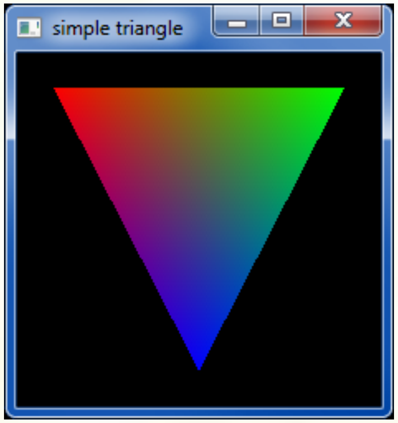
- Continuing the OpenGL basic pipeline

OpenGL API Example

```
glShadeModel(GL_SMOOTH);  // smooth color interpolation
glEnable(GL_DEPTH_TEST);  // enable hidden surface removal

glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glBegin(GL_TRIANGLES);  // every 3 vertexes makes a triangle
  glColor4ub(255, 0, 0, 255);    // RGBA=(1,0,0,100%)
  glVertex3f(-0.8,  0.8,  0.3);  // XYZ=(-8/10,8/10,3/10)

  glColor4ub(0, 255, 0, 255);    // RGBA=(0,1,0,100%)
  glVertex3f( 0.8,  0.8, -0.2);  // XYZ=(8/10,8/10,-2/10)

  glColor4ub(0, 0, 255, 255);    // RGBA=(0,0,1,100%)
  glVertex3f( 0.0, -0.8, -0.2);  // XYZ=(0,-8/10,-2/10)
glEnd();
```



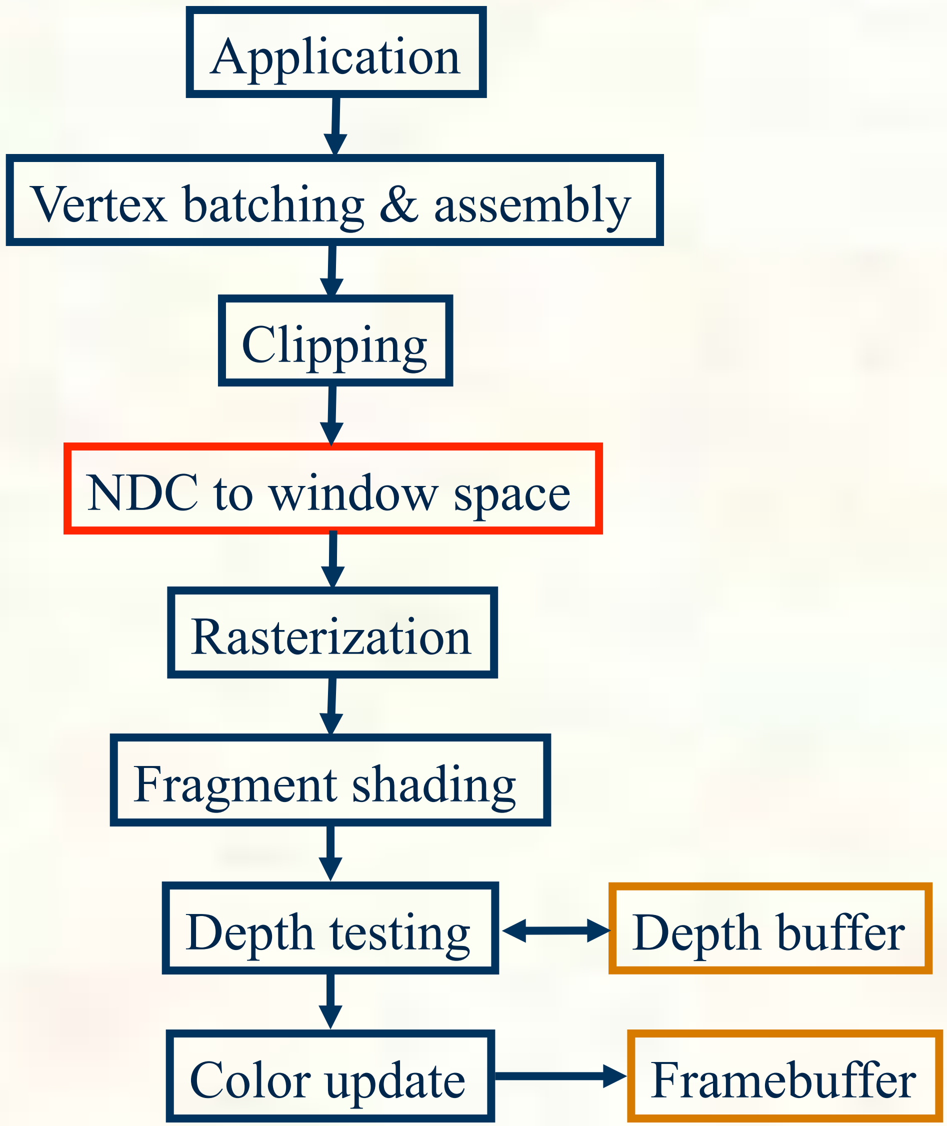
GLUT API Example




```
#include <GL/glut.h>  // includes necessary OpenGL headers

void display() {
  // << insert code on prior slide here >>
  glutSwapBuffers();
}
void main(int argc, char **argv) {
  // request double-buffered color window with depth buffer
  glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
  glutInit(&argc, argv);
  glutCreateWindow("simple triangle");
  glutDisplayFunc(display); // function to render window
  glutMainLoop();
}
```

NDC to Window Space

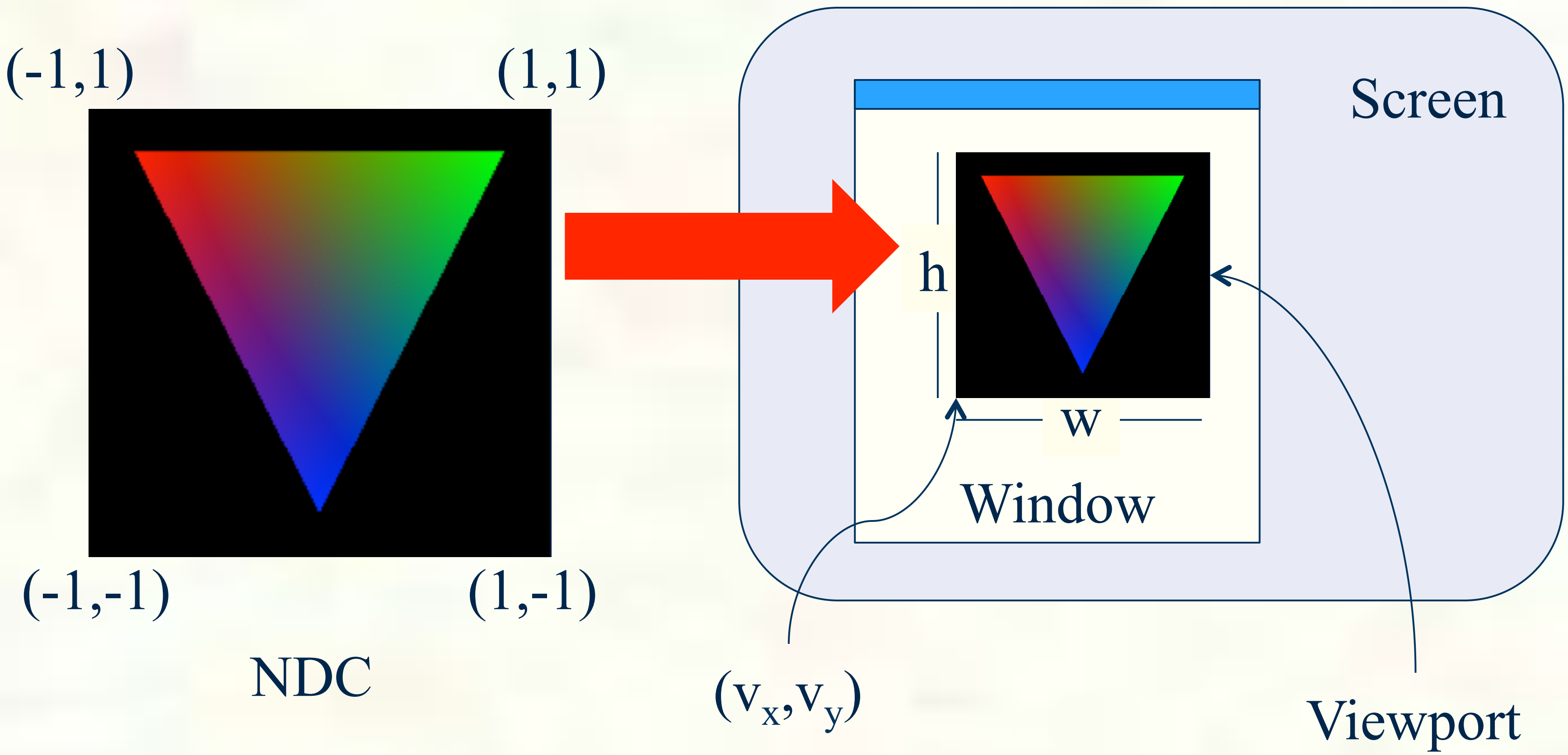
- NDC is "normalized" to the $[-1,+1]^3$ cube
  - Nice for clipping
  - But doesn't yet map to pixels on the screen
- **Next:** a transform from NDC space to window space



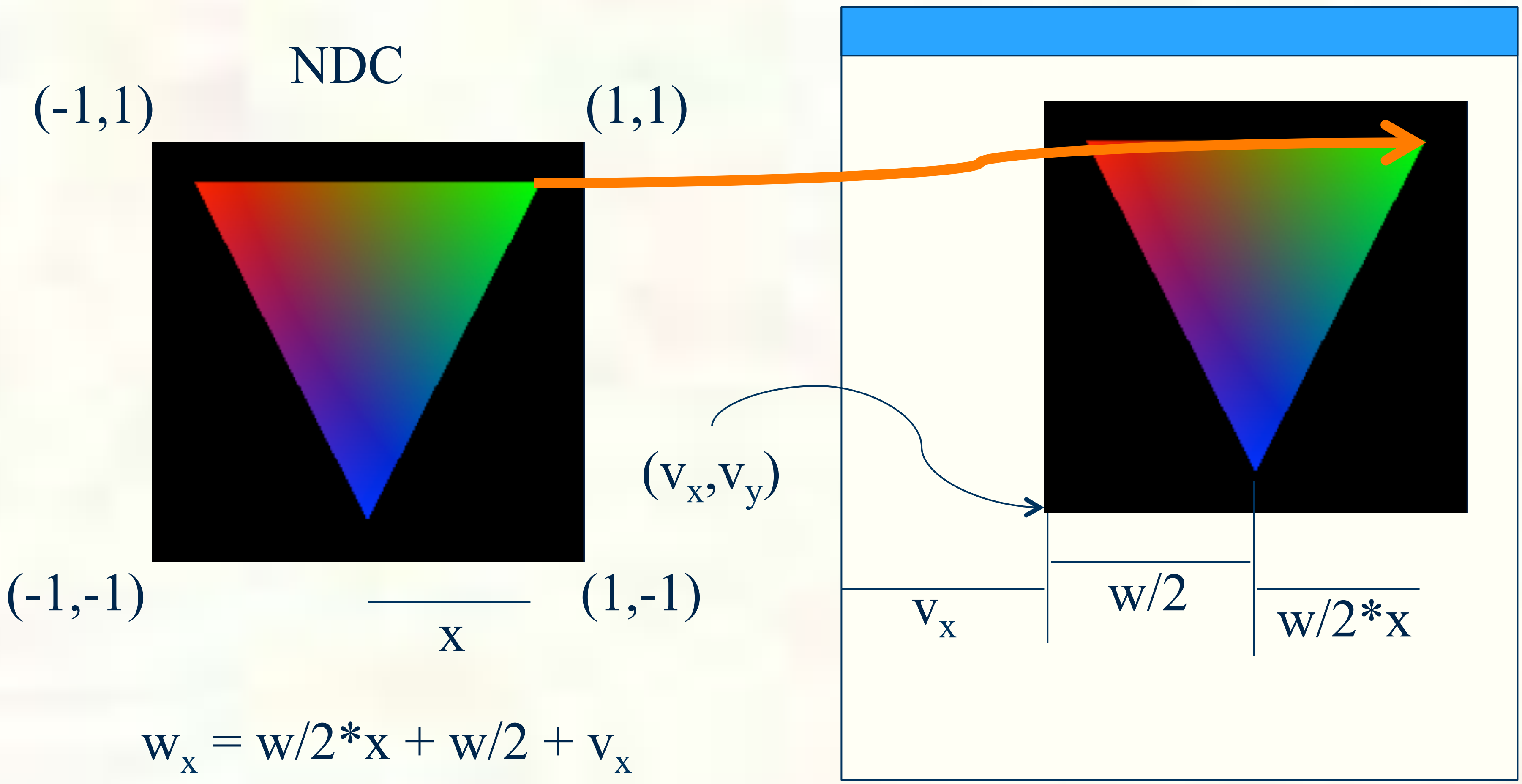
Viewport and Depth Range

- OpenGL has 2 commands to configure the state to map NDC space to window space
  - glViewport(GLint vx, GLint vy, GLsizei w, GLsizei h);
    - Typically programmed to the window's width and height for *w* & *h* and zero for both *vx* & *vy*
    - **Example:** glViewport(0, 0, window_width, window_height);
  - glDepthRange(GLclampd n, GLclampd f);
    - *n* for near depth value, *f* for far depth value
    - Normally set to glDepthRange(0,1)
      - Which is an OpenGL context's initial depth range state
- The mapping from NDC space to window space depends on *vx*, *vy*, *w*, *h*, *n*, and *f*

Viewport Transform



Viewport Transform



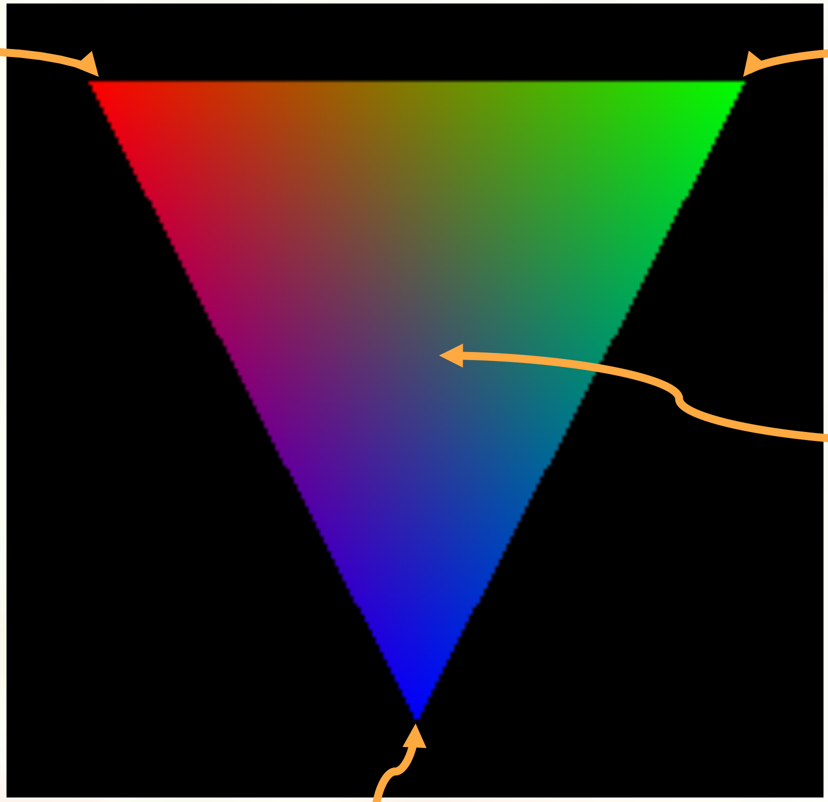
$$w_x = w/2 * x + w/2 + v_x$$
$$w_y = h/2 * y + h/2 + v_y$$

Mapping NDC to Window Space

- Assume (x,y,z) is the NDC coordinate that's passed to glVertex3f in our simple_triangle example
- Location in viewport (window space) is
  - $w_x = (w/2)*x + v_x + w/2$
  - $w_y = (h/2)*y + v_y + h/2$

Transforming Vertices

- Assume glViewport(0,0,500,500) has been called

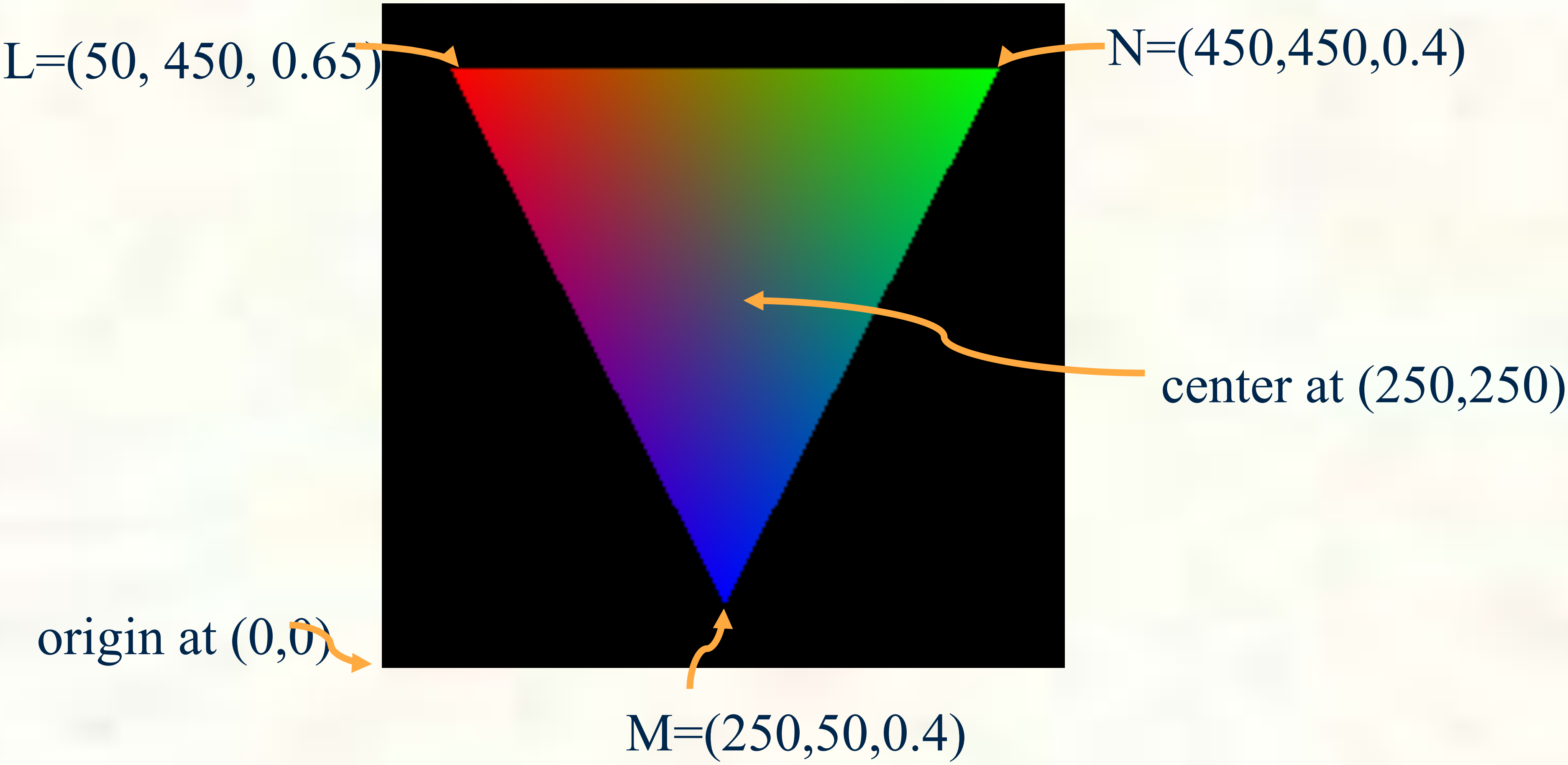


Apply the Transforms

- First vertex :: (-0.8, 0.8, 0.3)
  - $w_x = (w/2)*x + v_x + w/2 = 250*(-0.8) + 250 = 50$
  - $w_y = (h/2)*y + v_y + h/2 = 250*(0.8) + 250 = 450$
- Second vertex :: (0.8, 0.8, -0.2)
  - $w_x = (w/2)*x + v_x + w/2 = 250*(-0.8) + 250 = 50$
  - $w_y = (h/2)*y + v_y + h/2 = 250*(0.8) + 250 = 450$
- Third vertex :: (0, -0.8, -0.2)
  - $w_x = (w/2)*x + v_x + w/2 = 250*0 + 250 = 250$
  - $w_y = (h/2)*y + v_y + h/2 = 250*(-0.8) + 250 = 50$

Window Space Coordinates

- Assume glViewport(0,0,500,500) has been called



Where is glViewport set?

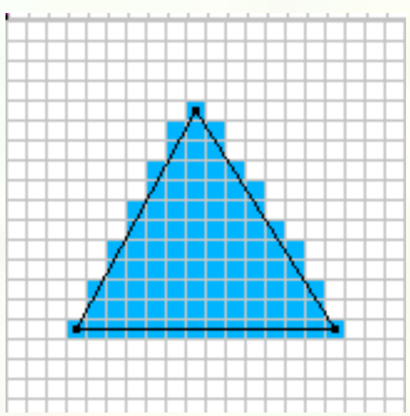
- The simple_triangle program never calls glViewport
  - That's OK because GLUT will call glViewport for you if you don't register your own per-window callback to handle when a window is reshaped (resized)
  - Without a reshape callback registered, GLUT will simply call glViewport(0, 0, window_width, window_height);
- Alternatively, you can use glReshapeFunc to register a callback
  - Then calling glViewport or otherwise tracking the window height becomes your application's responsibility
  - Example reshape callback:
    ```
    void reshape(int w, int h) {
      glViewport(0, 0, w, h);
    }
    ```
  - Example registering a reshape callback:
    glReshapeFunc(reshape);
- **FYI**: OpenGL maintains a lower-left window-space origin
  - Whereas most 2D graphics APIs use upper-left

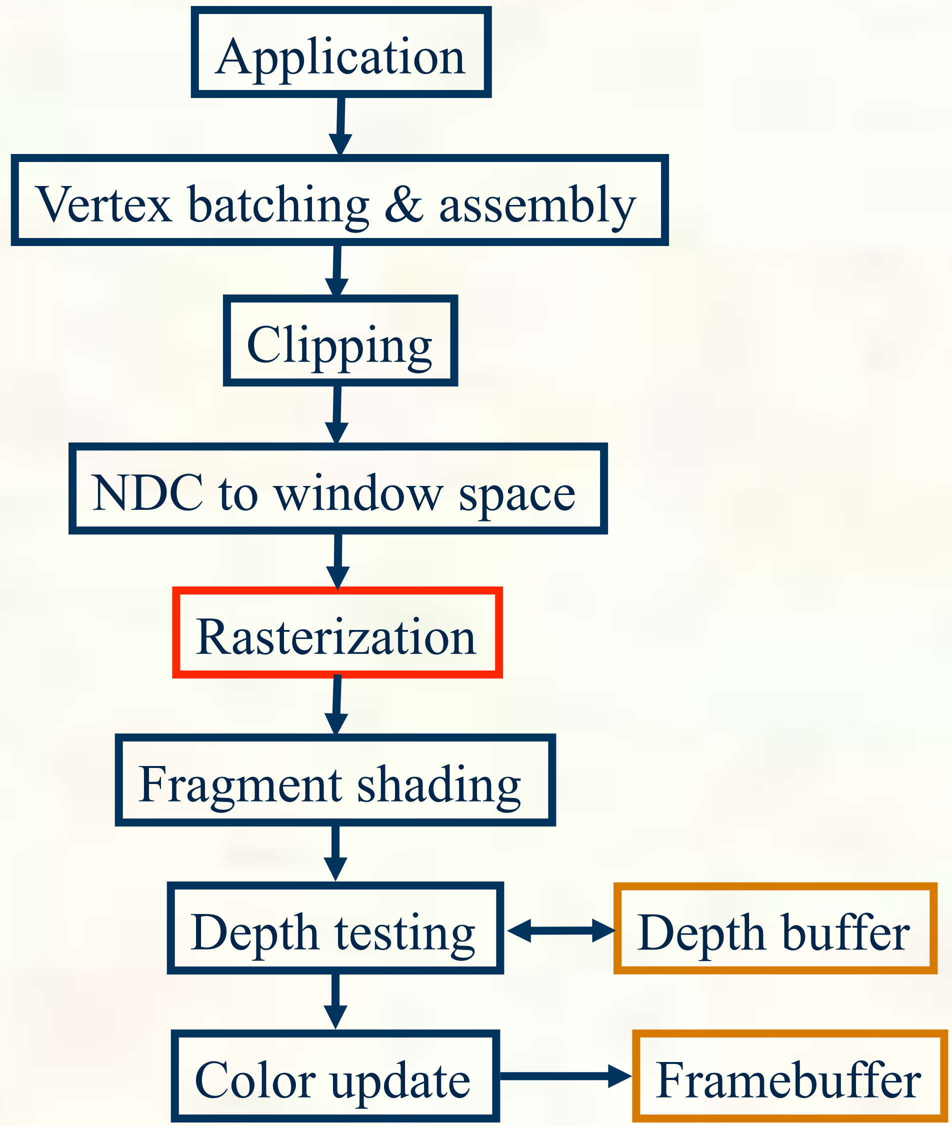
What about glDepthRange?

- Simple applications don't normally need to call glDepthRange
  - Notice the simple_triangle program never calls glDepthRange
- Rationale
  - The initial depth range of [0,1] is fine for most application
  - It says the entire available depth buffer range should be used
- When the depth range is [0,1] the equation for window-space z simplifies to $wz = \frac{1}{2} \times z + \frac{1}{2}$

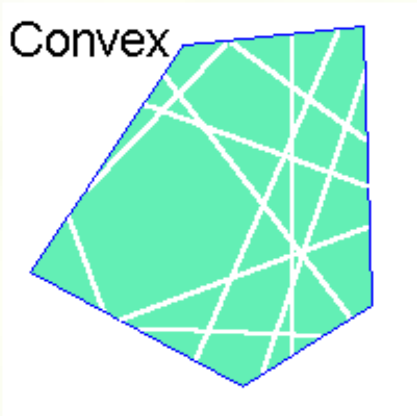
Rasterization

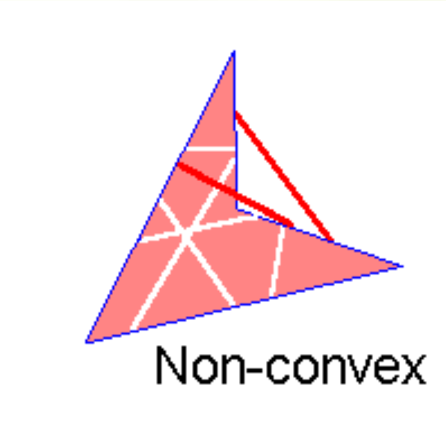
- Process of converting a clipped triangle into a set of sample locations covered by the triangle
  - Also can rasterize points and lines





Concave vs. Convex

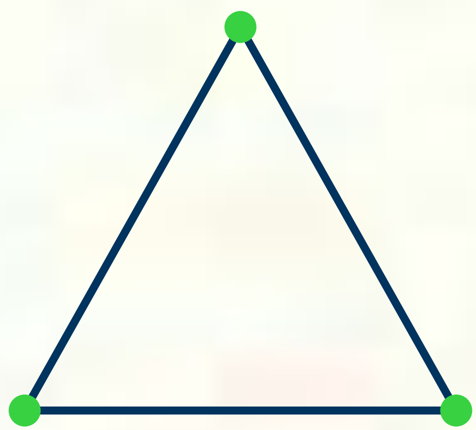


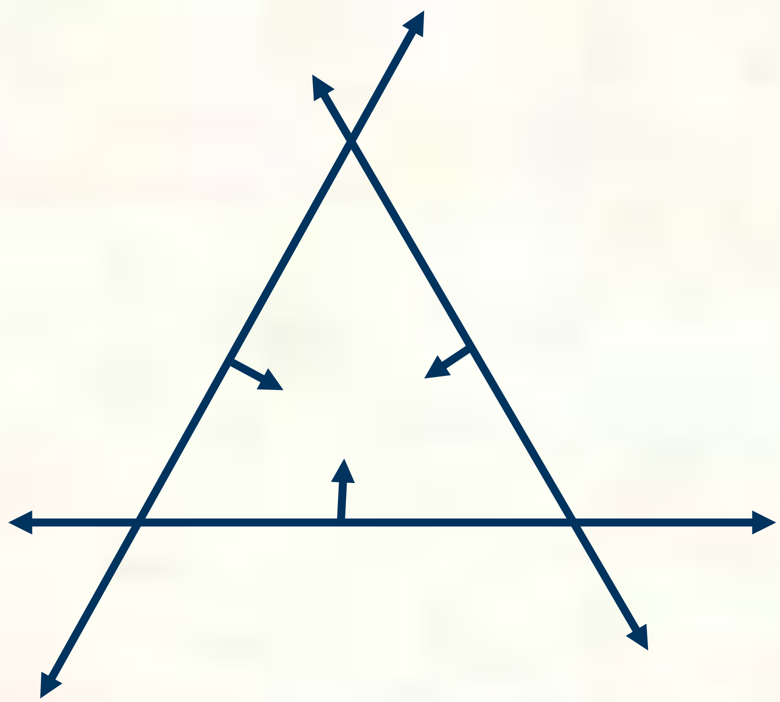


- Region is convex if any two points can be connected by a line segment where all points on this segment are also in the region
  - Opposite is non-convex
- Concave means the region is connected but NOT convex
  - Connected means there's some path (not necessarily a line) from every two points in the region that is entirely in the region

Determining a Triangle

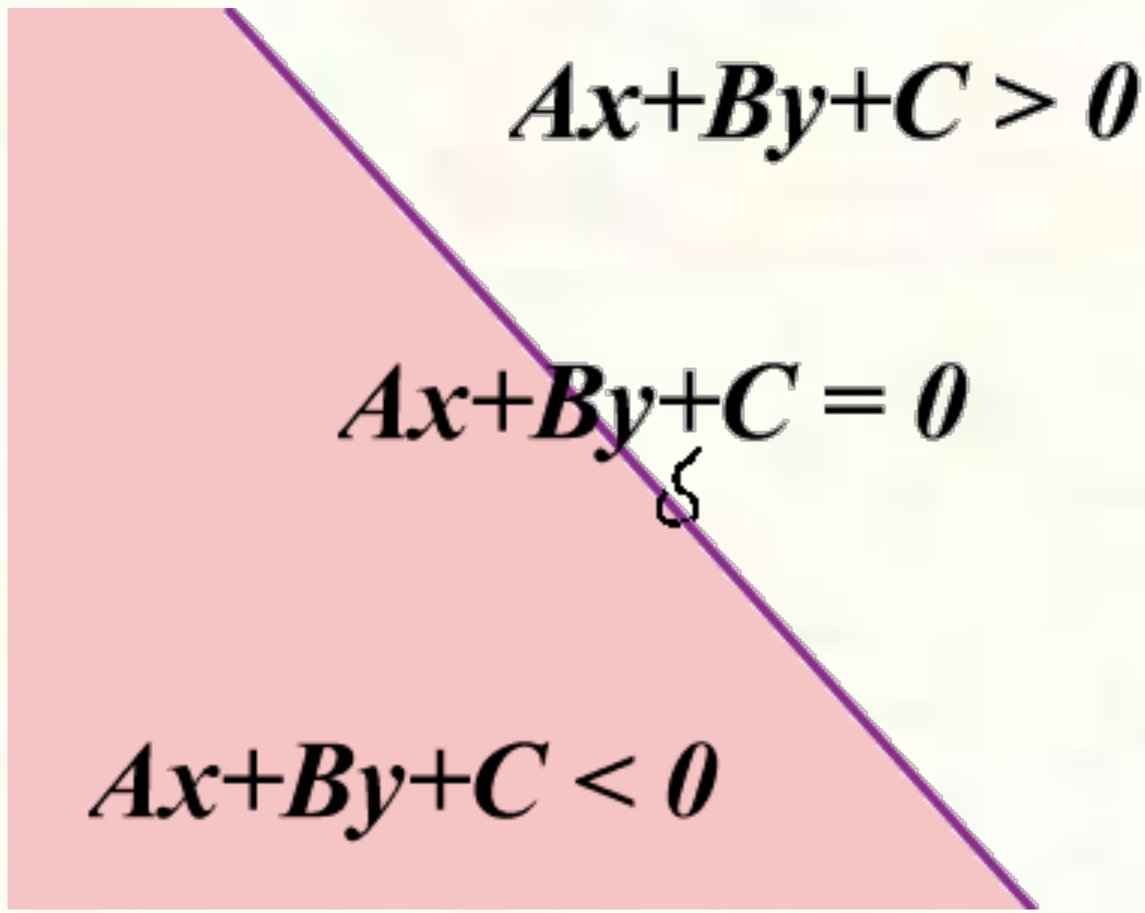
- **Classic view**: 3 points determine a triangle
  - Given 3 vertex positions, we determine a triangle
  - Hence glVertex3f/ glVertex3f/glVertex3f
- **Rasterization view**: 3 oriented edge equations determine a triangle





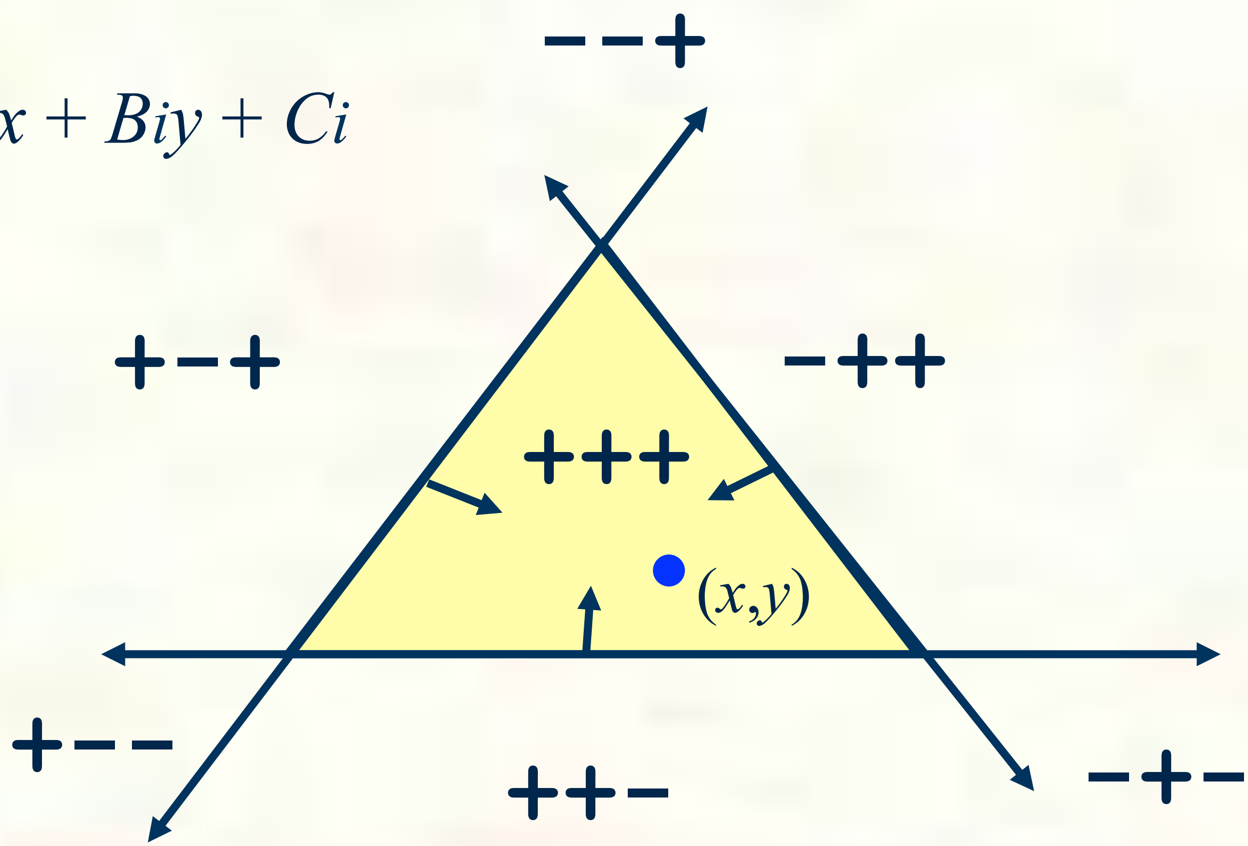
Each oriented edge equation in form:
$A*x + B*y + C \geq 0$

Oriented Edge Equations



7 Cases

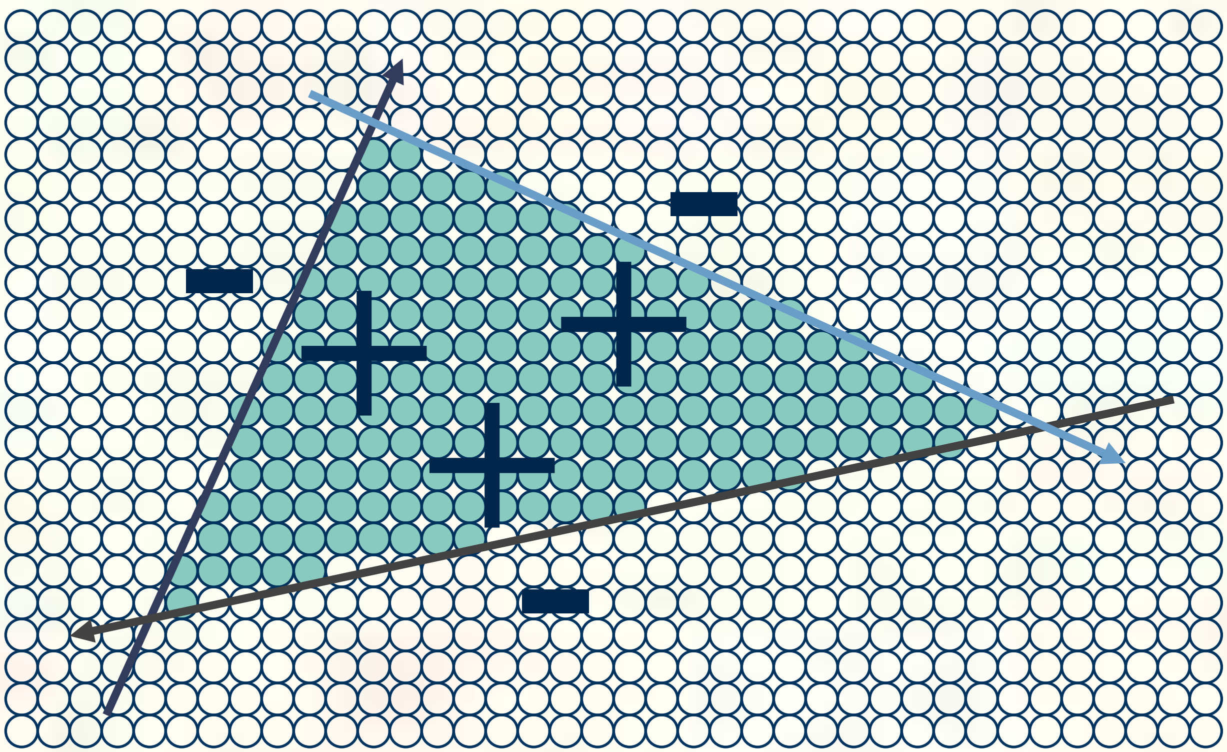
$$E_i(x,y) = Aix + Biy + Ci$$



Inside Triangle Test

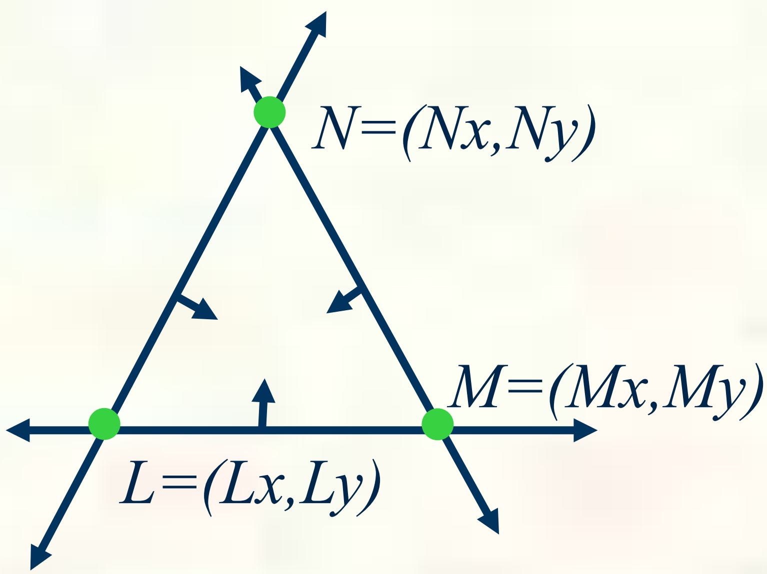
- Evaluate edge equations at grid of sample points
  - If sample position is "inside" all 3 edge equations, the position is "within" the triangle
  - Implicitly parallel—all samples can be tested at once

- Good for hardware implementation
  - Pixel-planes
  - Pineda tiled extension



Creating Edge Equations

- Triangle rasterization need edge equations
  - How do we make edge equations?
- An edge is a line so determined by two points
  - Each of the 3 triangle edges is determined by two of the 3 triangle vertexes (L, M, N)



How do we get

$$A*x + B*y + C \geq 0$$

for each edge from L, M, and N?

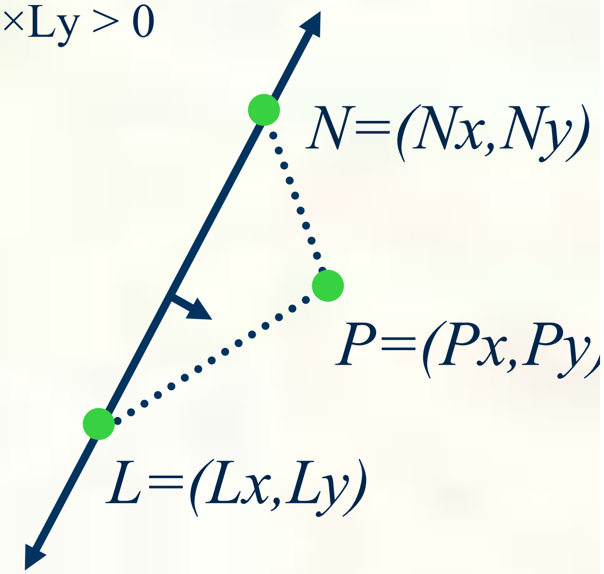
Edge Equation Setup

- How do you get the coefficients A, B, and C?
- Determinants help—consider the LN edge:

P is an arbitrary point

$$\begin{vmatrix} N_x - L_x & N_y - L_y \\ P_x - L_x & P_y - L_y \end{vmatrix} > 0 \quad \textit{or more succinctly} \quad \begin{vmatrix} N - L \\ P - L \end{vmatrix} > 0$$

- **Expansion:** $(Ly\text{-}Ny)\times Px + (Nx\text{-}Lx)\times Py + Ny\times Lx\text{-}Nx\times Ly > 0$
  - $A_{LN} = Ly\text{-}Ny$
  - $B_{LN} = Nx\text{-}Lx$
  - $C_{LN} = Ny\times Lx\text{-}Nx\times Ly$
- **Geometric interpretation:** twice signed area of the triangle LPN



Look at the LN edge

- **Expansion:**

  (Ly-Ny)×Px + (Nx-Lx)×Py + Ny×Lx-Nx×Ly > 0

  - A_{LN} = Ly-Ny = 450-450 = 0
  - B_{LN} = Nx-Lx = 50-450 = -400
  - C_{LN} = Ny×Lx-Nx×Ly = 180,000
- Is center at (250,250) in the triangle?
  - $A_{LN} \times 250 + B_{LN} \times 250 + C_{LN}$ = ???
  - $0 \times 250 - 400 \times 250 + 180{,}000 = 80{,}000$
    - 80,000 > 0 so (250,250) is <u>in</u> the triangle

All Three Edge Equations

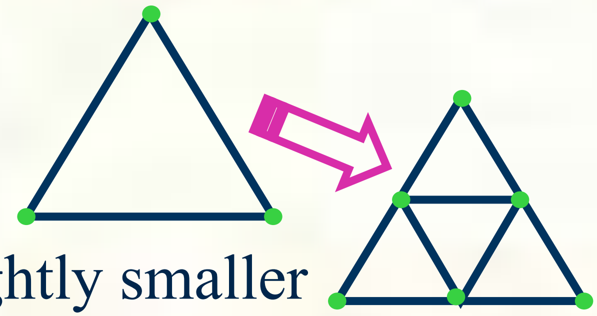
- All three triangle edge equations:

$$\begin{vmatrix} N-P \\ M-P \end{vmatrix} > 0 \qquad \begin{vmatrix} N-L \\ P-L \end{vmatrix} > 0 \qquad \begin{vmatrix} P-L \\ M-L \end{vmatrix} > 0$$

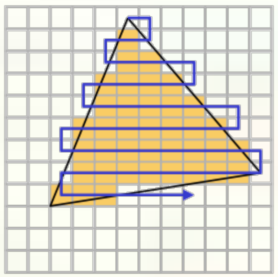
- Satisfy all 3 and P is in the triangle
  - And then rasterize at sample location P
- **Caveat:** if $\begin{vmatrix} N-L \\ M-L \end{vmatrix} < 0$ reverse the comparsion sense

Other Rasterization Approaches

- Subdivision approaches
  - Easy to split a triangle into 4 triangles
  - Keep splitting triangles until they are slightly smaller than your samples
    - Often called micro-polygon rendering
    - Chief advantage is being able to apply displacements during the subdivision

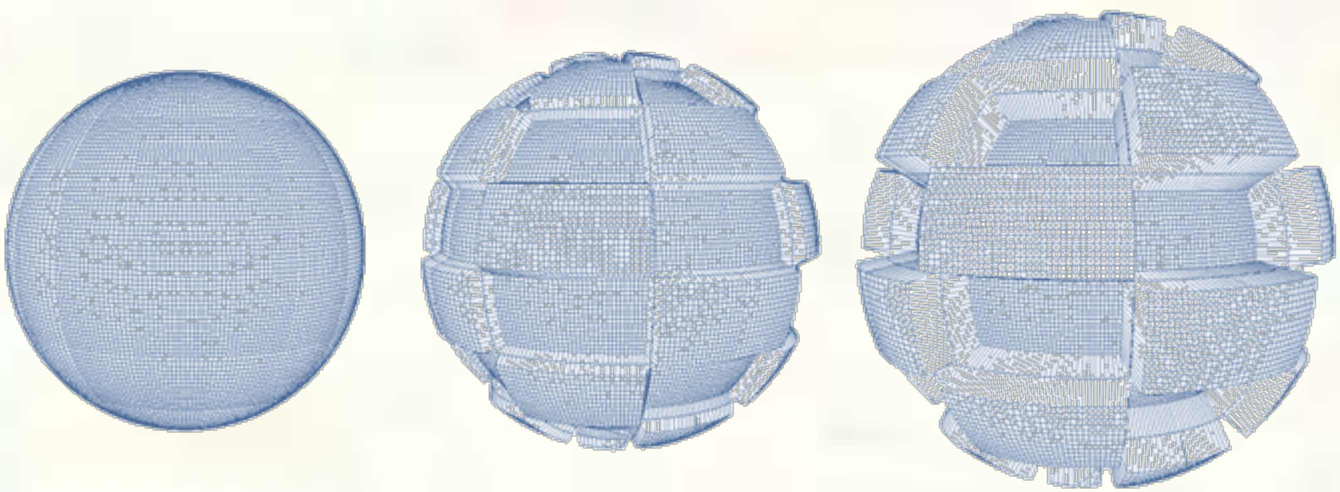


- Edge walking approaches
  - Often used by CPU-based rasterizers
  - Much more sequential than Pineda approach
  - Work efficient and amendable to fixed-point implementation



Micropolygons

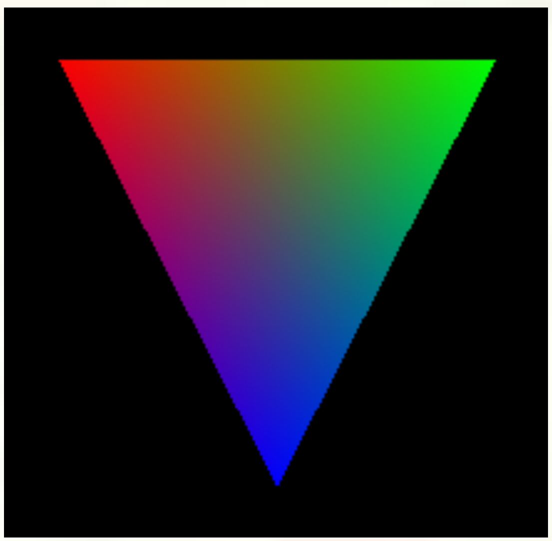
- Rasterization becomes a geometry dicing process
  - Approach taken by Pixar
    - For production rendering when scene detail and quality is at a premium; interactivity, not so much
  - High-level representation is generally patches rather than mere triangles

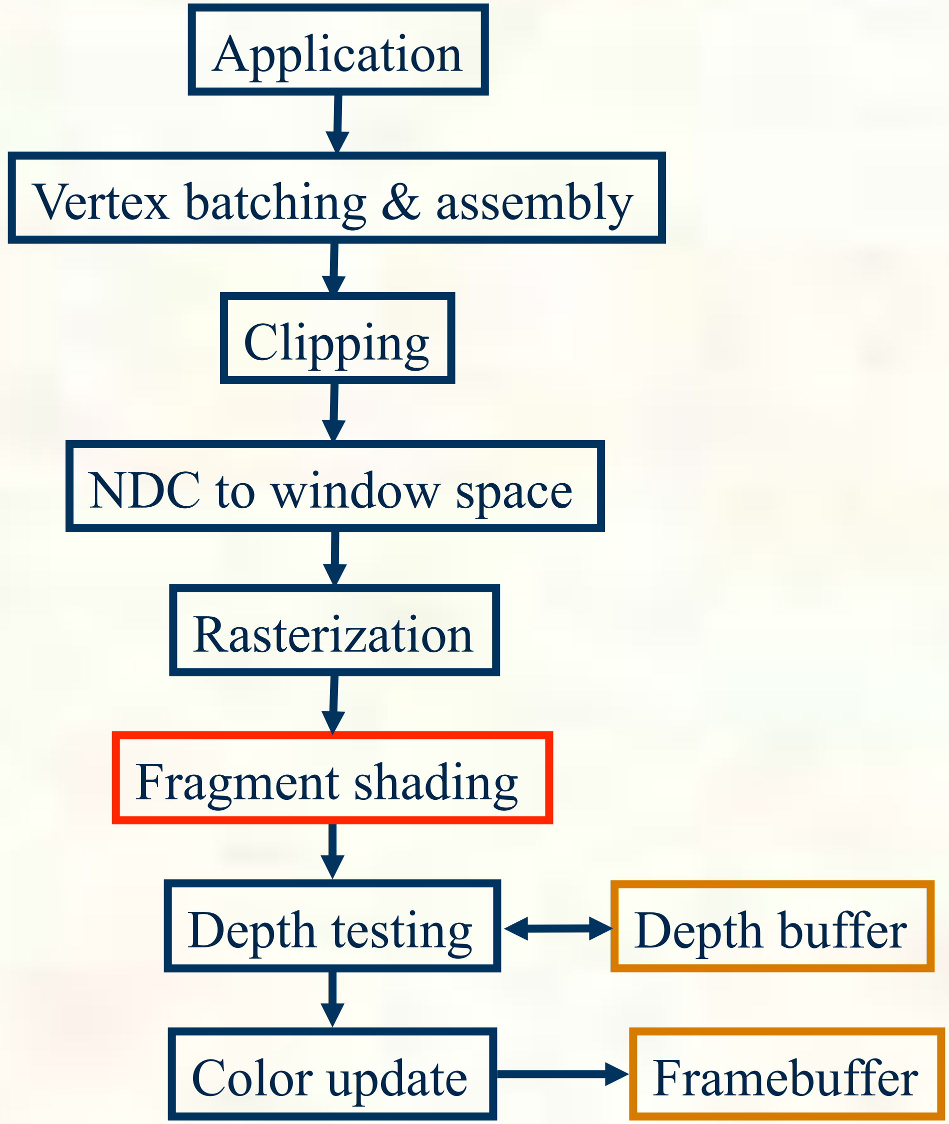


Displacement mapping of a meshed sphere [Pixar, RenderMan]

Simple Fragment Shading

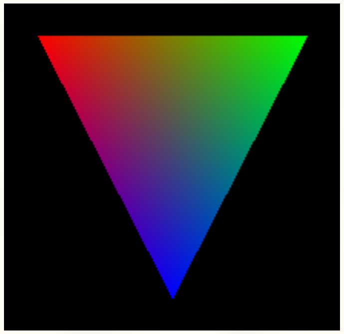
- For all samples (pixels) within the triangle, evaluate the interpolated color
  - Requires having math to determine color at the sample (x,y) location

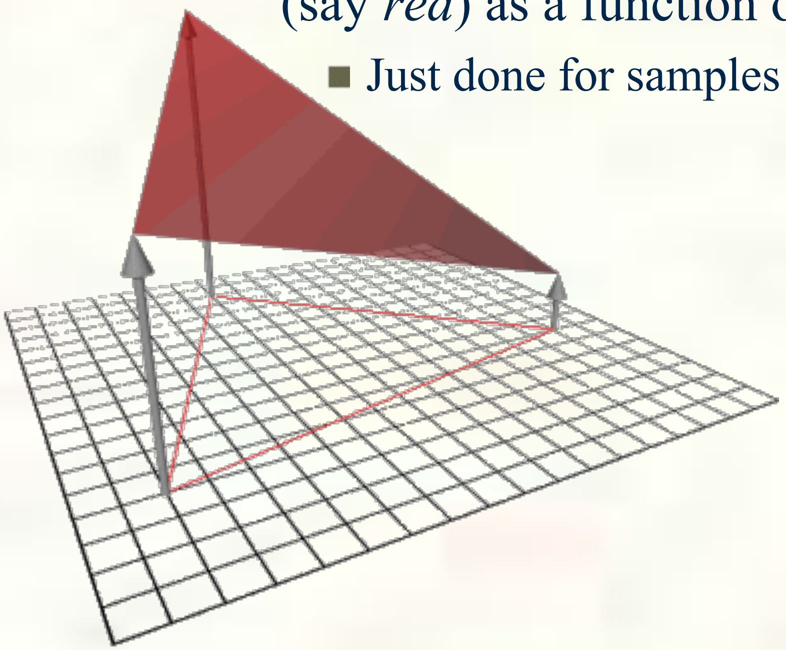




Color Interpolation

- Our simple triangle is drawn with smooth color interpolation
  - Recall: glShadeModel(GL_SMOOTH)
- How is color interpolated?
  - Think of a plane equation to computer each color component (say *red*) as a function of (x,y)
    - Just done for samples positions within the triangle





$$"redness" = A_{red}x + B_{red}y + C_{red}$$

Setup Plane Equation

- Setup plane equation to solve for "red" as a function of (x,y)

Setup system of equations

$$\begin{bmatrix} L_{red} \\ M_{red} \\ N_{red} \end{bmatrix} = \begin{bmatrix} L_x & L_y & 1 \\ M_x & M_y & 1 \\ N_x & N_y & 1 \end{bmatrix} \begin{bmatrix} A_{red} \\ B_{red} \\ C_{red} \end{bmatrix}$$

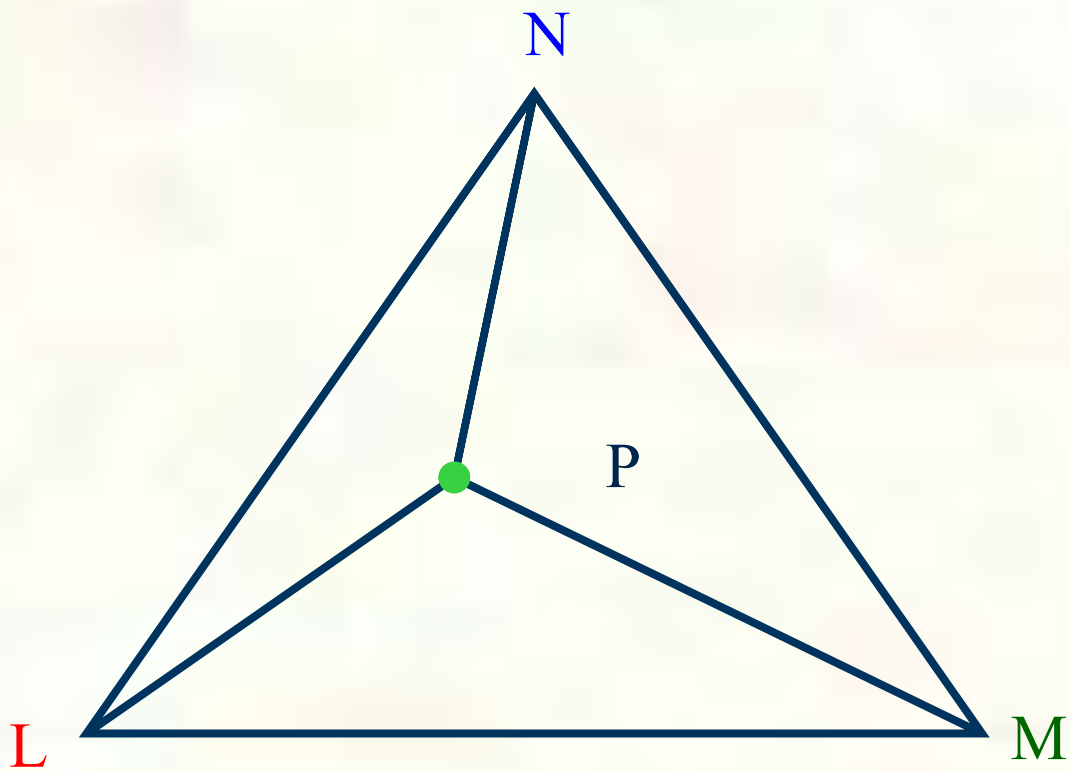
Solve for plane equation coefficients A, B, C

$$\begin{bmatrix} L_x & L_y & 1 \\ M_x & M_y & 1 \\ N_x & N_y & 1 \end{bmatrix}^{-1} \begin{bmatrix} L_{red} \\ M_{red} \\ N_{red} \end{bmatrix} = \begin{bmatrix} A_{red} \\ B_{red} \\ C_{red} \end{bmatrix}$$

Do the same for green, blue, and alpha (opacity)…

More Intuitive Way to Interpolate

- Barycentric coordinates



$$\frac{\text{Area}(PMN)}{\text{Area}(LMN)} = \alpha$$

$$\frac{\text{Area}(LPN)}{\text{Area}(LMN)} = \beta$$

$$\frac{\text{Area}(LMP)}{\text{Area}(LMN)} = \gamma$$

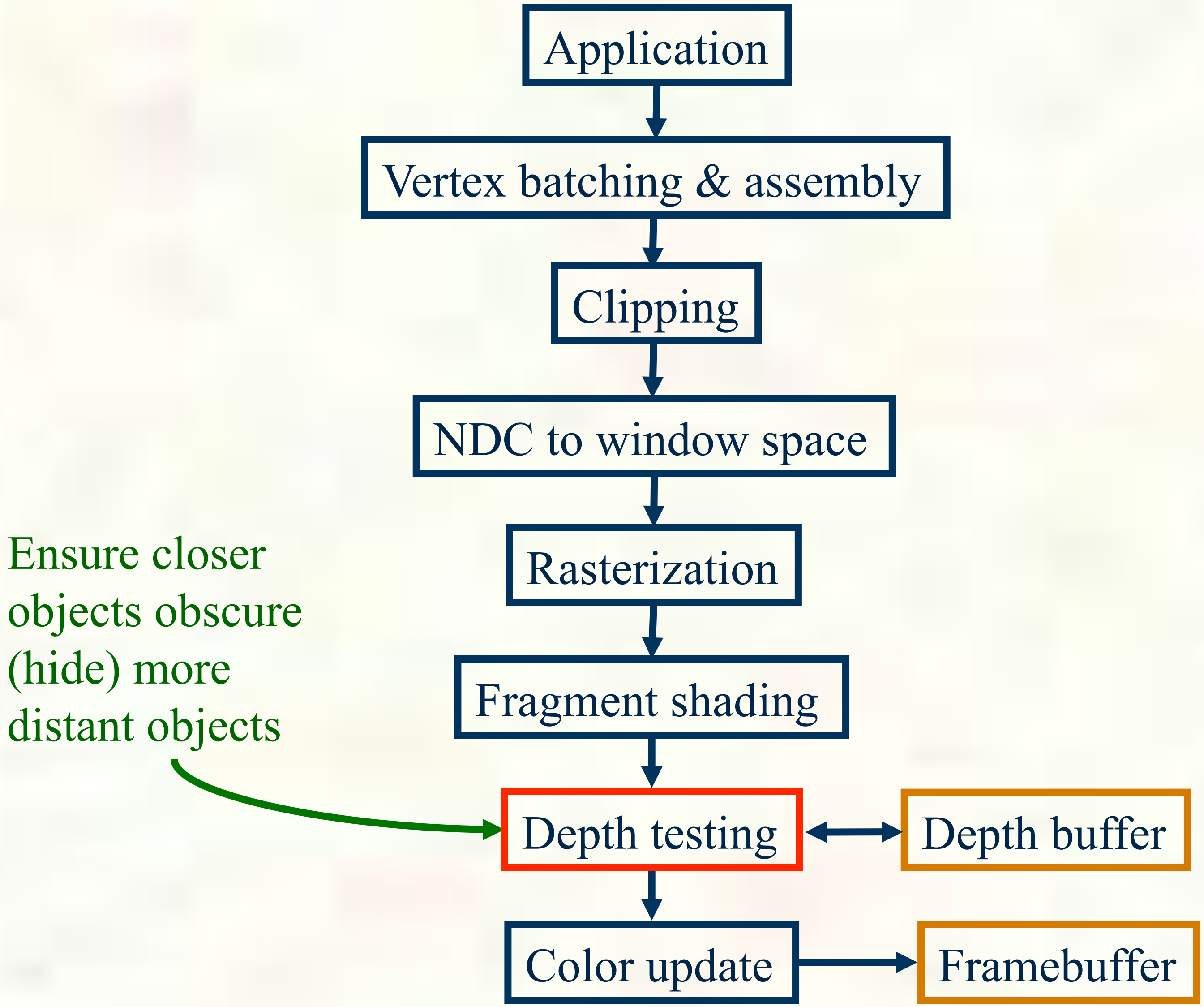
Note: $\alpha + \beta + \gamma = 0$ by construction

$$\text{attribute}(P) = \alpha\times\text{attribute}(L) + \beta\times\text{attribute}(M) + \gamma\times\text{attribute}(N)$$

Hardware Triangle Rendering Rates

- Top GPUs can setup over a billion triangles per second for rasterization
- Triangle setup & rasterization is just one of the (many, many) computation steps in GPU rendering

A Simplified Graphics Pipeline



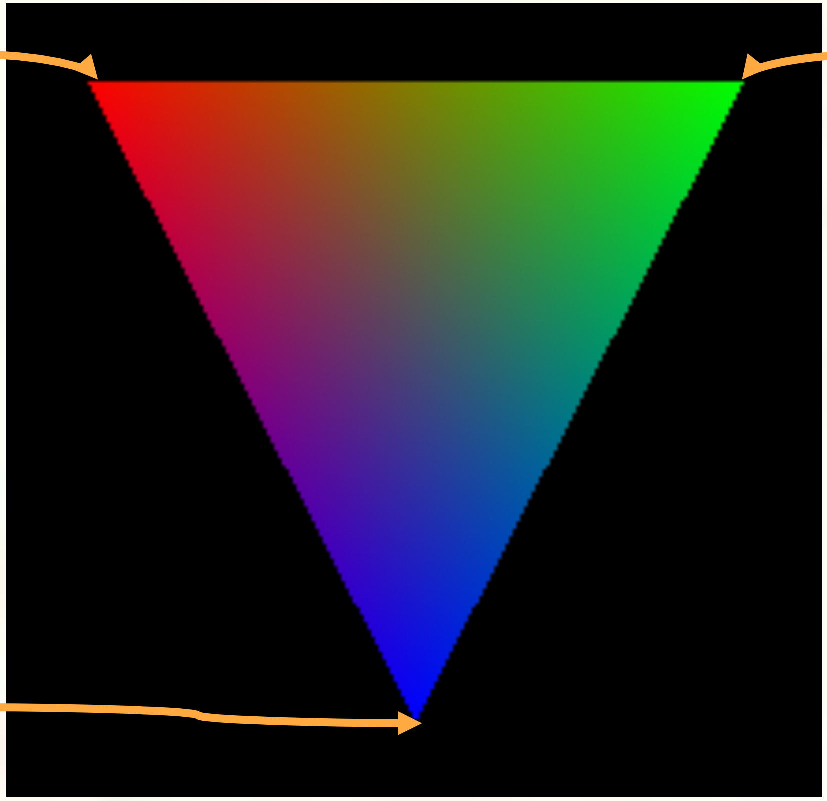
Interpolating Window Space Z

- Plane equation coefficients (A, B, C) generated by multiplying inverse matrix by vector of per-vertex attributes

$$\begin{bmatrix} L_x & L_y & 1 \\ M_x & M_y & 1 \\ N_x & N_y & 1 \end{bmatrix}^{-1} \begin{bmatrix} L_z \\ M_z \\ N_z \end{bmatrix} = \begin{bmatrix} A_z \\ B_z \\ C_z \end{bmatrix}$$

Simple Triangle Vertex Depth

- Assume glViewport(0,0,500,500) has been called
  - And glDepthRange(0,1)



$L_z = 0.65$

$M_z = 0.40$

$N_z = 0.40$

Interpolating Window Space Z

- Substitute per-vertex (x,y) and Z values for the L, M, and N vertexes

$$\begin{bmatrix} 50 & 450 & 1 \\ 250 & 50 & 1 \\ 450 & 450 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0.65 \\ 0.4 \\ 0.4 \end{bmatrix} = \begin{bmatrix} A_z \\ B_z \\ C_z \end{bmatrix}$$

A_z= -0.000625

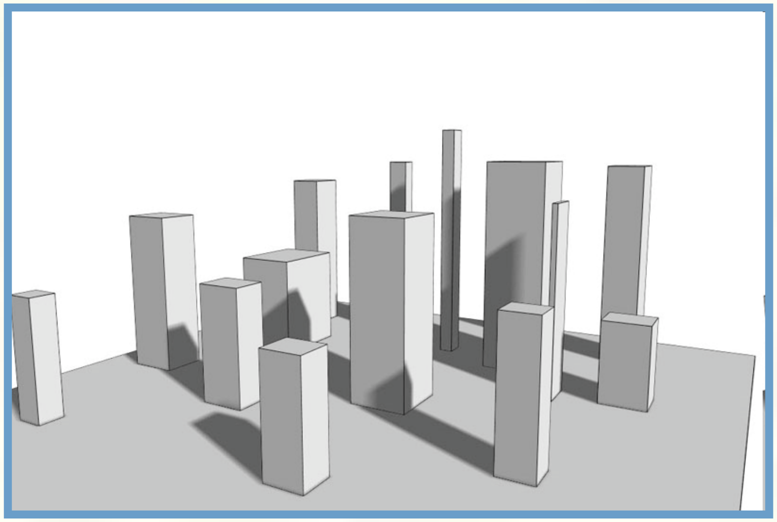
B_z = 0.0003125

C_z = 0.540625

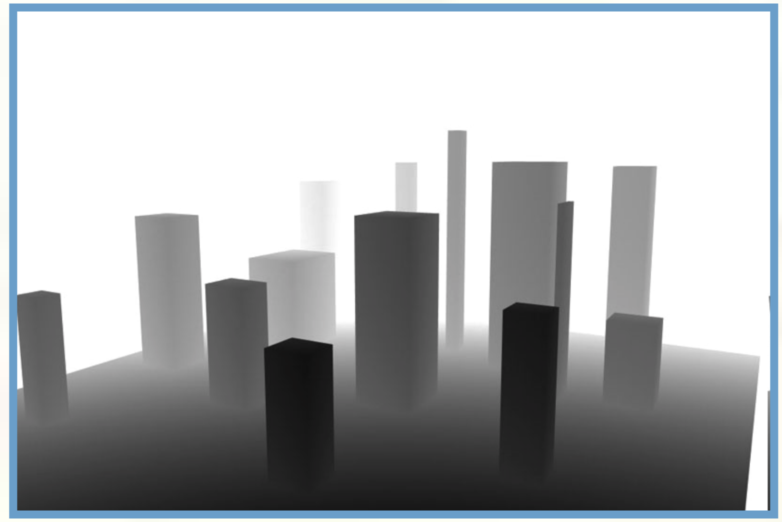
Complete Z plane equation

Z(x,y) = -0.000625*x + 0.0003125*y + 0.540625

Depth Buffer Visualized



Depth-tested
3D scene



Z or depth values
white = 1.0 (far), black = 0.0 (near)

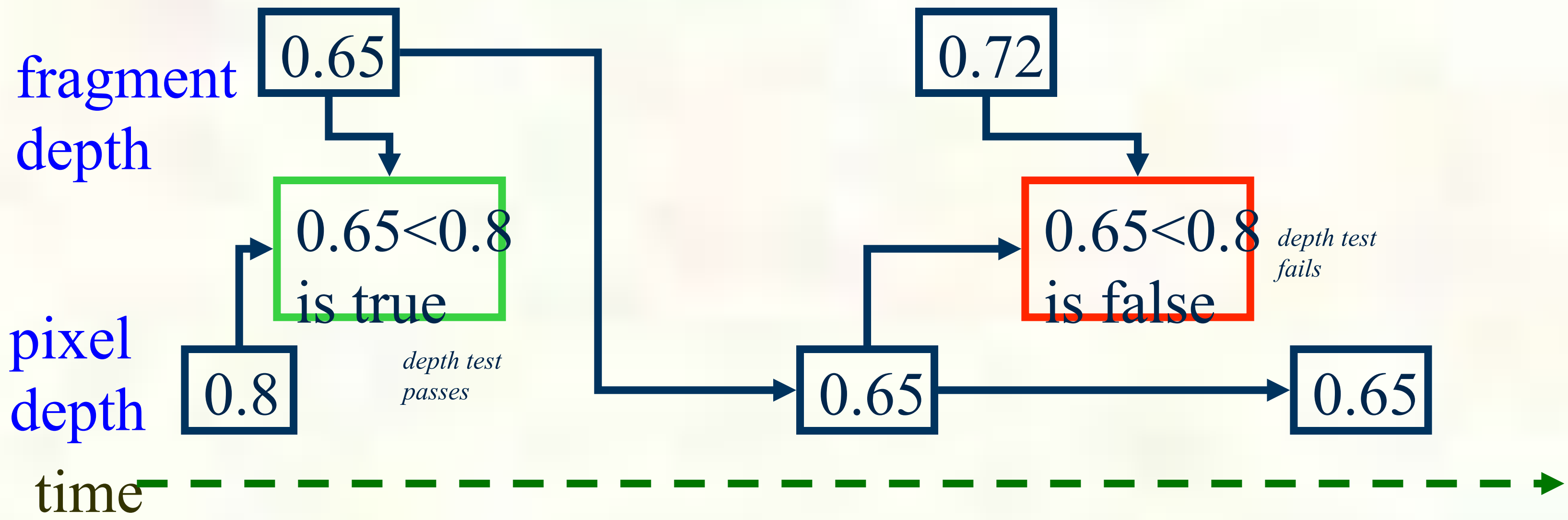
Depth Buffer Algorithm

- Simple, brute force
  - Every color sample in framebuffer has corresponding depth sample
  - Discrete, solves occlusion in pixel space
  - Memory intensive, but fast for hardware
- Basic algorithm
  - Clear the depth buffer to its "maximum far" value (generally 1.0)
  - Interpolate fragment's Z
  - Read fragment's corresponding depth buffer sample Z value
  - If interpolated Z is less than (closer) than Z from depth buffer
    - Then replace the depth buffer Z with the fragment's Z
      - And also allow the fragment's shaded color to update the corresponding color value in color buffer
    - Otherwise discard fragment
      - Do <u>not</u> update depth or color buffer

Depth Buffer Example

- Fragment gets rasterized
- Fragment's Z value is interpolated
  - Resulting Z value is 0.65
- Read the corresponding pixel's Z value
  - Reads the value 0.8
- Evaluate depth function
  - 0.65 GL_LESS 0.8 is **true**
  - So 0.65 replaces 0.8 in the depth buffer
- Second primitive rasterizes same pixel
- Fragment's Z value is interpolated
  - Resulting Z value is 0.72
- Read the corresponding pixel's Z value
  - Reads the value 0.65
- Evaluate depth function
  - 0.72 GL_LESS 0.65 is **false**
  - So the fragment's depth value and color value are discarded

Depth Test Operation

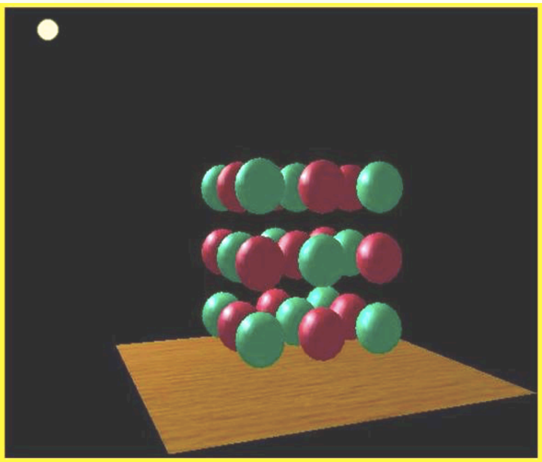


OpenGL API for Depth Testing

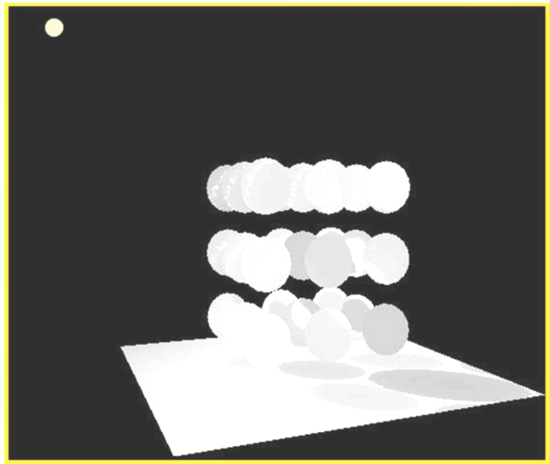
- Simple to use
  - Most applications just "enable" depth testing and hidden surfaces are removed
  - Enable it: glEnable(GL_DEPTH_TEST)
    - Disabled by default
    - **<u>Must</u>** have depth buffer allocated for it to work
      - **Example:** glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH)
- More control
  - Clearing the depth buffer
    - glClear(GL_DEPTH_BUFFER_BIT | otherBits)
    - glClearDepth(*zvalue*)
      - Initial value is 1.0, the maximum Z value in the depth buffer
  - glDepthFunc(*<u>zfunc</u>*)
    - *zfunc* is one of GL_LESS, GL_GREATER, GL_EQUAL, GL_GEQUAL, GL_LEQUAL, GL_ALWAYS, GL_NEVER, GL_NOTEQUAL
    - Initial value is GL_LESS
  - glDepthMask(*boolean*)
    - True means write depth value if depth test passes; if false, don't write
    - Initial value is GL_TRUE
  - glDepthRange
    - Maps NDC Z values to window-space Z values
    - Initially [0,1], mapping to the entire available depth range

Not Just for View Occlusion

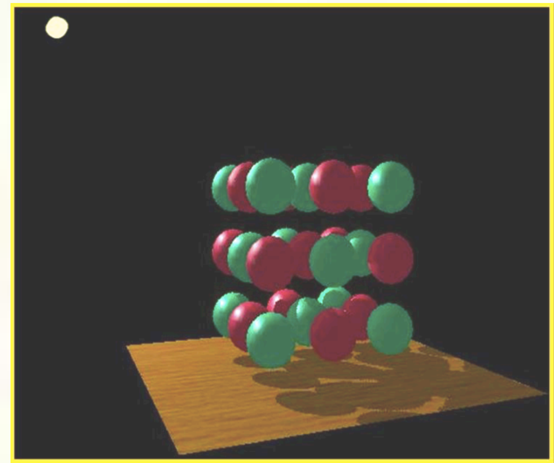
Depth Buffers also Useful for Shadow Generation



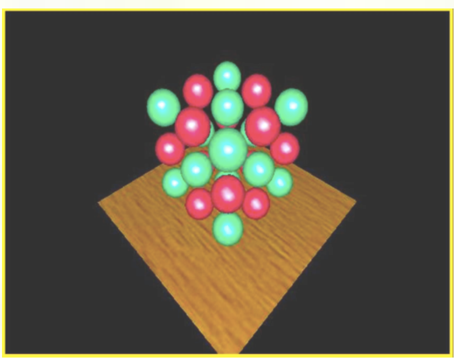
Without Shadows



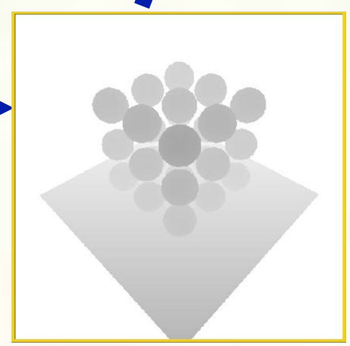
Projected Shadow Map



With Shadows

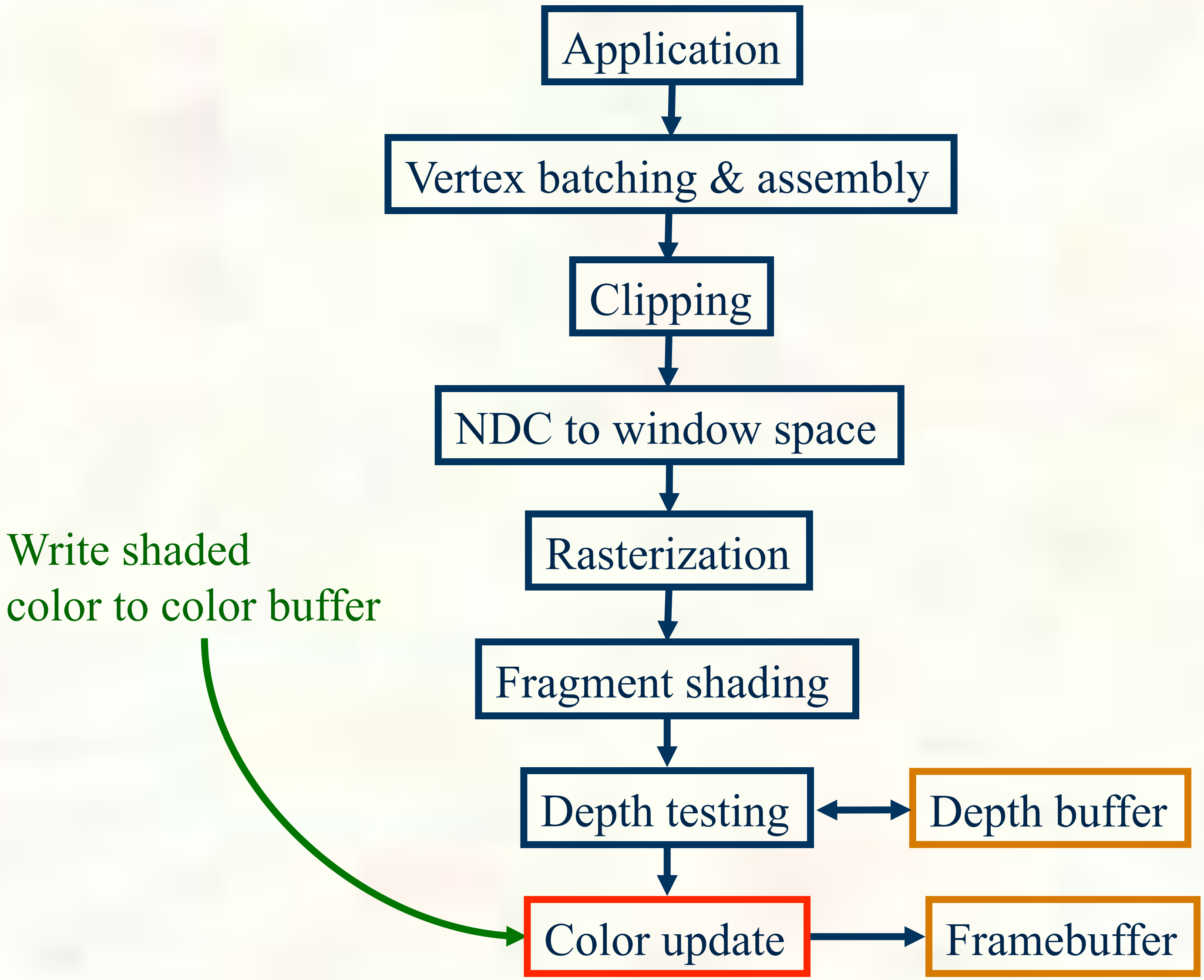


Light’s View



Light’s View Depth

A Simplified Graphics Pipeline



Next Lecture

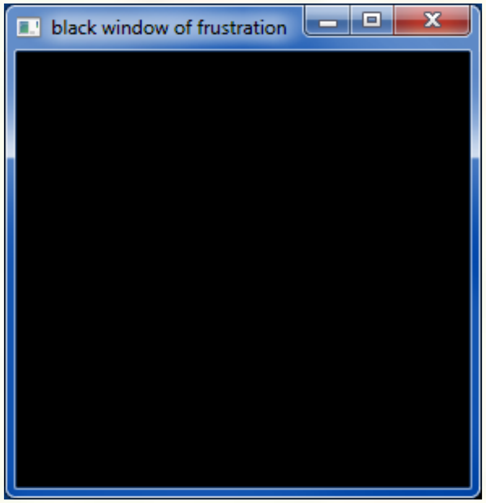
- Graphics Math, Transforms
  - *Interpolation, vector math, and number representations for computer graphics*

Next Lecture

- Finish OpenGL pipeline
- Transforms and Graphics Math
  - *Interpolation, vector math, and number representations for computer graphics*

Programming tips

- 3D graphics, whether OpenGL or Direct3D or any other API, can be frustrating
  - You write a bunch of code and the result is



Nothing but black window; where did your rendering go??

Things to Try

- Set your clear color to something other than black!
  - It is easy to draw things black accidentally so don't make black the clear color
  - But black is the initial clear color
- Did you draw something for one frame, but the next frame draws nothing?
  - Are you using depth buffering? Did you forget to clear the depth buffer?
- Remember there are near and far clip planes so clipping in Z, not just X & Y
- Have you checked for glGetError?
  - Call glGetError once per frame while debugging so you can see errors that occur
  - For release code, take out the glGetError calls
- Not sure what state you are in?
  - Use glGetIntegerv or glGetFloatv or other query functions to make sure that OpenGL's state is what you think it is
- Use glutSwapBuffers to flush your rendering and show to the visible window
  - Likewise glFinish makes sure all pending commands have finished
- Try reading
  - http://www.slideshare.net/Mark_Kilgard/avoiding-19-common-opengl-pitfalls
  - This is well worth the time wasted debugging a problem that could be avoided

Thanks

- Presentation approach and figures from
  - David Luebke [2003]
  - Brandon Lloyd [2007]
  - *Geometric Algebra for Computer Science* [Dorst, Fontijne, Mann]
  - via Mark Kilgard