55. List, in order, the transformations that must be carried out to produce object coordinates, world coordinates, eye coordinates, clip coordinates, normalized device coordinates, and window coordinates.

Object Coordinates = Modeling Matrix \* Object Vector

World Coordinates = View Matrix \* Object Coordinates

Eye Coordinates = Setting the View Matrix to the Identity Matrix, the output object coordinates produce the eye coordinates

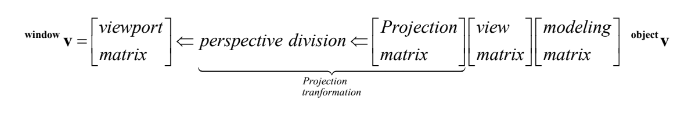
Clip Coordinates = Projection Matrix applied to the Eye Coordinates

Normalized Device Coordinates = Apply the Clipping processes using a graphics algorithm to the clip coordinates to get these.

Window Coordinates = Applying Perspective Division to the Normalized Device Coordinates to get the Window vector of coordinates

56. Given matrices that describe modeling, viewing, projection, and viewport transformations,

order them correctly for use in transforming a vertex from Object to Window coordinates.



57. Given a modeling transformation, write an expression of statement that would correctly

transform a normal vector to World coordinates.

Object Coordinates = Modeling Matrix \* Object Vector

World Coordinates = View Matrix \* Object Coordinates

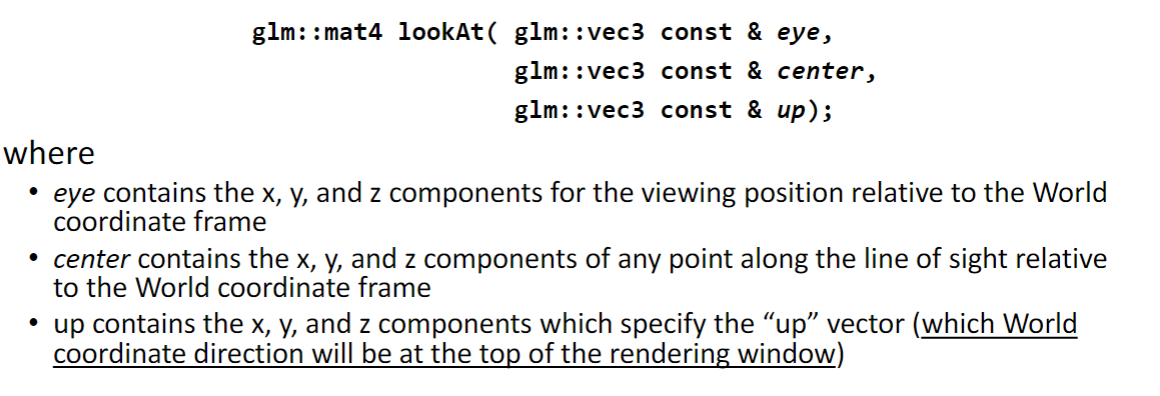
**Chapter Seven (Viewing, Projection, and Viewport Transformations)**

35. Given a description of the position and orientation of a view point in world coordinates, write a code fragment containing calls to the GLM translate and rotate functions that creates the composite transformation needed to produce the desired viewing transformation.

Same operations as we conduct when moving objects, but this time we are modifying a viewpoint as opposed to a collection of objects and setting it equal to viewingTransformation

36. Given a description of a position and orientation of a view point in world coordinates, use

GLM lookat to produce the desired viewing transformation.



37. Given a matrix that describes a viewing transformation use the matrix to transform a given

vertex from *world coordinates* to *eye coordinates*.

Find the inverse of the viewing transformation matrix, this gives us the eye coordinate frame.

EyeCoordinateFrame \* World Position Vector = Eye Vector

38. Describe and name the shape of the view volume associated with a perspective projection.

It is a called a frustum, which looks like truncated pyramid (A pyramid with the top cut off)

39. Describe and name of the shape of the view volume associated with an orthogonal

projection.

This is called a view volume, and it is shaped like a box

40. Define clipping and state the purpose of a clipping plane.

Clipping is the act of not rendering pieces of an object that are not in our view frame, which leads to severely increased performance. A clipping plane allows us to use certain graphics algorithms to determine if a piece of an object is in frame or not. If it’s borders exceed the clipping plane, that chunk is removed

41. Use a simple projection matrix and *perspective division* to transform a given vertex from *eye*

*coordinates* to *clip coordinates*.

First: Projection Matrix \* eye Coordinates;

Second: Take the result of that and divide by the Z component

42. Describe what relation is required between the aspect ratios associated with the projection

view volume and the viewport in order to produce an undistorted view of a “scene”.

Aspect Ratios of the view volume and the viewport need to be equal

43. Given a matrix that describes the viewing transformation, write an expression or a code

fragment that gives the location of the view point relative to World coordinates.

Viewing Transformation Matrix \* View Point

**Chapter Eight (Graphics Pipeline Algorithms)**

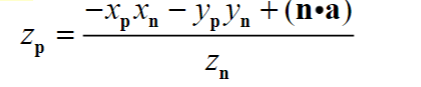
44. Given three vertices that describe a triangle and a vector that describes the viewing direction,

determine whether or not the triangle faces the viewpoint.

If the vector pointing out of the triangle is the opposite that of the one pointing from the viewpoint, the triangle is seen. Otherwise, it is not.

45. Given the description of plane that includes a normal vector and a point on the plane,

determine if a given vertex is in “front” or “behind” the plane.

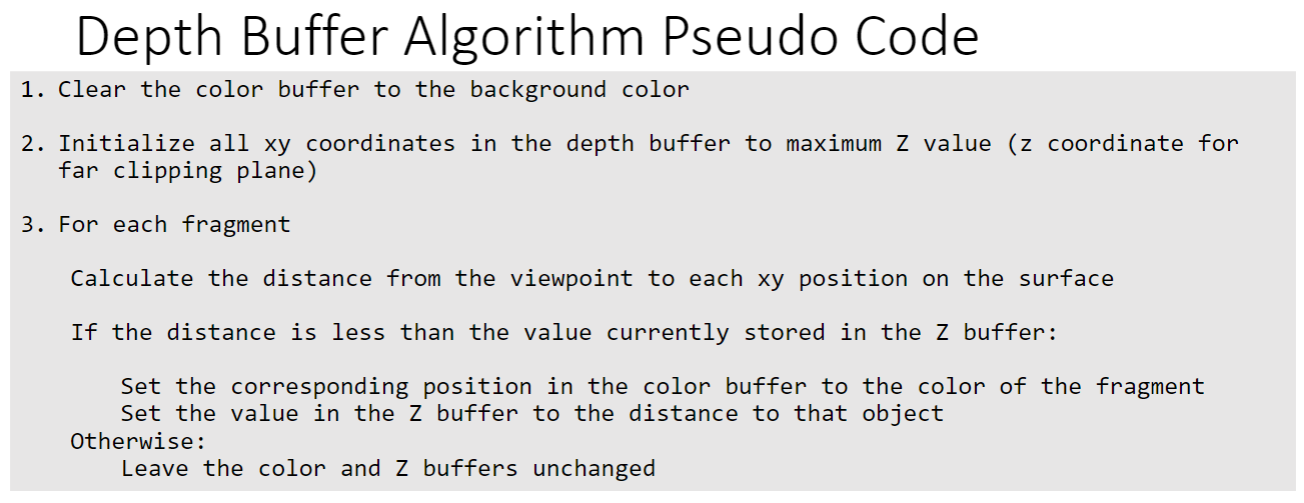


46. Write a code fragment that correctly implements the *depth test* and how it is used to control

writing to the color and depth buffers.

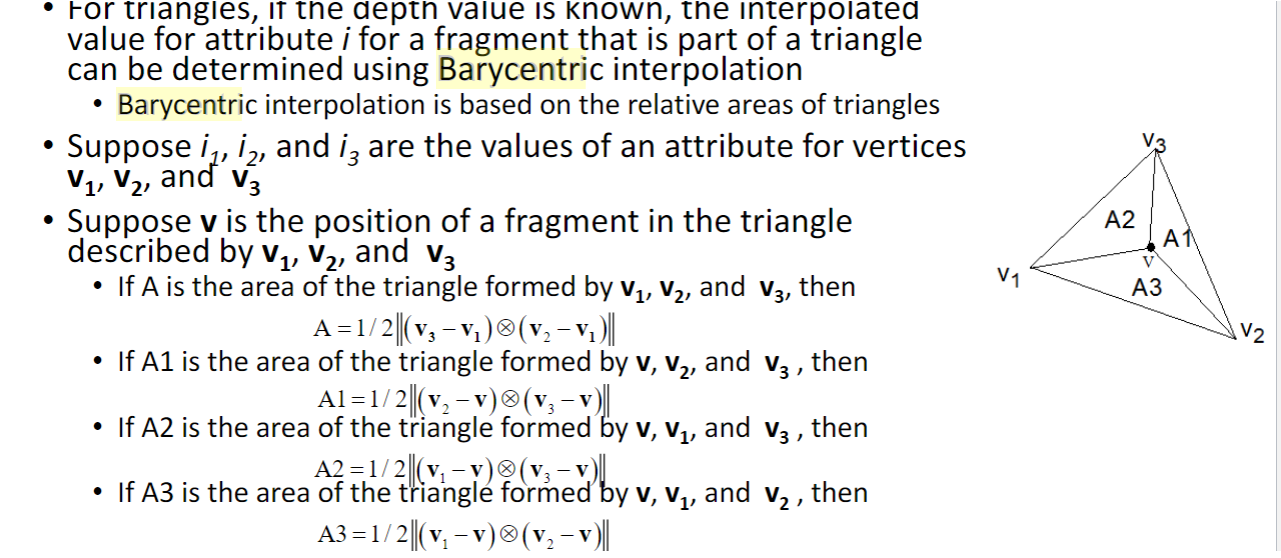
A depth test calculates the difference in z values between the view point and the object pixel we are currently looking at. We use the result of this to compare what is already in the depth buffer, and if the object is closer, we change what is stored in the color buffer.

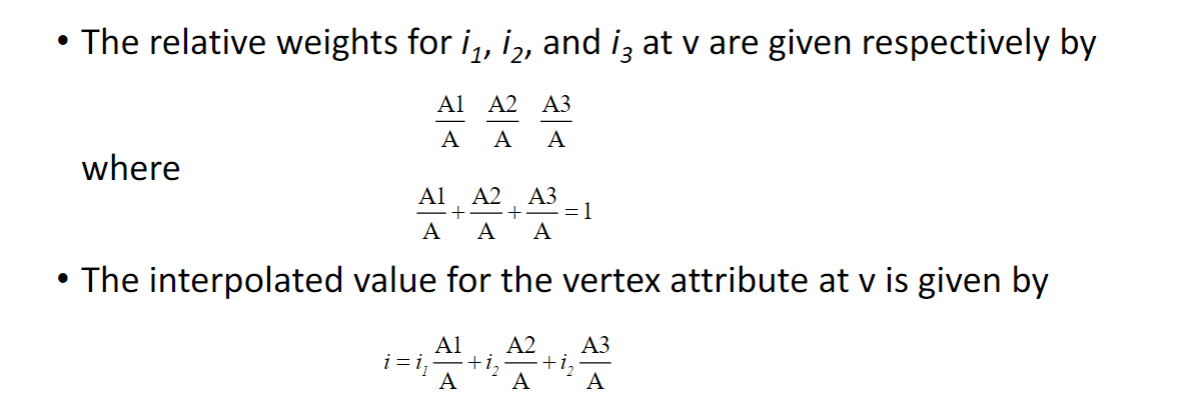
47. Write a correct pseudo code description of the depth buffer algorithm.



48. Use the areas of triangles to perform Barycentric interpolation of vertex attributes for a point

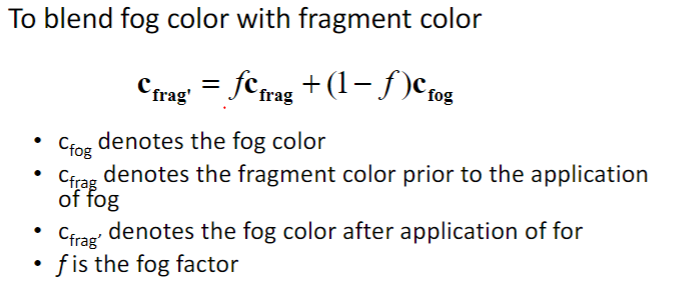
on the interior of the triangle.





**Fog and Alpha Blending**

49. Given the values parameters associated with linear, exponential, or exponential 2 fog, the location of the view point, and the fragment color and location, calculate the red, green, and/or blue components that will be written into the color buffer.



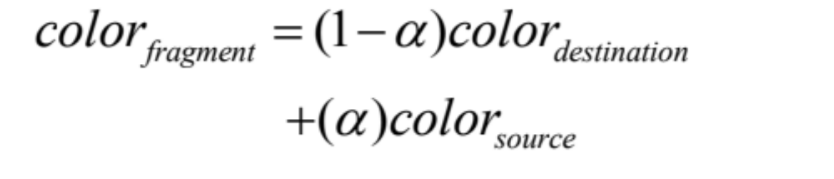
50. Define the terms *source* and *destination* as they relate to alpha blending.

Source is the color of the object we are currently attempting to add to the color buffer. Destination is the color that is already present in the buffer

51. Given the red, green, blue, and alpha components of two colors, use (source) alpha blending

to blend together the two colors.

alpha



52. Given the location of the viewpoint in World coordinates, state the necessary rendering order

for a group or opaque and transparent objects to correctly simulate transparency with alpha blending.

Render all Opaque objects prior to rendering transparent objects