Graphics Pipeline

Justin Huffman, Eryn Kelsey-Adkins, Jesus Torres

November 30, 2018

Abstract

The rendering pipeline is the series of steps a graphics system needs to follow to render a 3D scene to a 2D screen. These steps start with the Application, then the Geometry, the Rasterization, and finally the Display. This presentation provides a brief overview of each of the steps and the matrices used to transform 3D objects into 2D displays.

1 Introduction

3D graphics have become ubiquitous in the modern world. They are used in everything from movies to advertisements to, of course, computer games. Graphics applications like Blender and OpenGL make designing 3D graphics a relatively simply process, because they automate the transformations necessary to create and manipulate a 3D image and project it onto a 2D display.

The graphics pipeline is that series of steps needed to render a 3D scene to a 2D screen. [1]

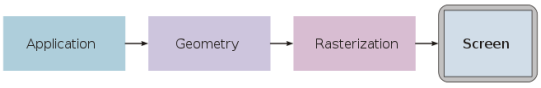

Figure 1: By PaterMcFly / Vierge Marie

This presentation provides a broad overview of these steps, focusing on the use of matrices in each transformation.

2 Application: how 3D models are stored

2.1 World Coordinate System

The world coordinates are the bedrock of the 3D model. This universal coordinate system is independent of the camera, and objects or primatives, which

[1] `https://commons.wikimedia.org/w/index.php?curid=58106609`

each have their own coordinate system, are placed within the world coordinates. Every point is defined by three vertices (x, y, z). The cartesian coordinates can be left- or right- handed, defined by whether the z vertex points into or out of the display.[2] Depending on the coordinate system being used, the models may need to be inverted when loading for veiwing.

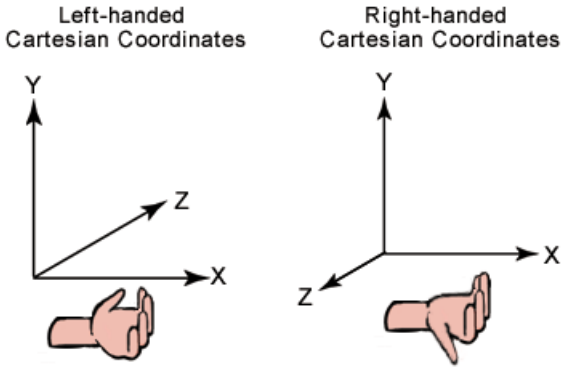

Figure 2: Left- and Right- Handed Cartesian Coordinates

2.2 Homogeneous Coordinates

Instead of operating in R^3, most graphics pipelines use homogeneous coordinates, which exist in R^4 and are of the form (x, y, z, w). Homogeneous Coordinates are useful because they allow the same transformations to be applied to both vertices (points in space that change when translated) and vectors (directions and facings, which do not).

A simple metaphor is to imagine a 2D projector. The X and Y values are easy to see, but what happens to the image as you move the projector closer or further away from the screen? The image is scaled up or down. This is similar to the w value in homogeneous coordinates.[3]

For now, we will set $w = 1$ for all vertices in our vertex list. If we had any vectors (i.e. normal vectors), we would set $w = 0$. (Other values of W will become relevant later in the projection step.) This will allow us to use the same standard $4x4$ matrices to transform both vertices and vectors (should we need them for more advanced geometry calculations).

2.3 Object Coordinate System

As mentioned above, each object has its own set of coordinates. Objects are most often represented by polyhedra, further broken down into triangles. Each triangle is defined, possibly counterintuitively, by 4 vertices: the 3 edges, and

[2] `cs.uic.edu`

[3] `https://www.tomdalling.com/blog/modern-opengl/explaining-homogenous-coordinates-and-projective-geometry/`

a fourth vertex defining the front or face of the primitive. When objects are moved in relation to the world coordinates, the origin of the object (in the object's coordinate system) is moved, with all other points transformed an equal amount.

Objects can further have a hierarchal coordinate system. The University of Illinois uses the example of a hand being positioned relative to the arm, rather than to the world coordinate system.

2.3.1 Polygon Mesh

To build the Object Coordinate System, objects are represented by a polygon mesh: the vertices, edges, and faces that define the shape of the object. Faces consist of simple convex polygons to simplify rendering.[4]

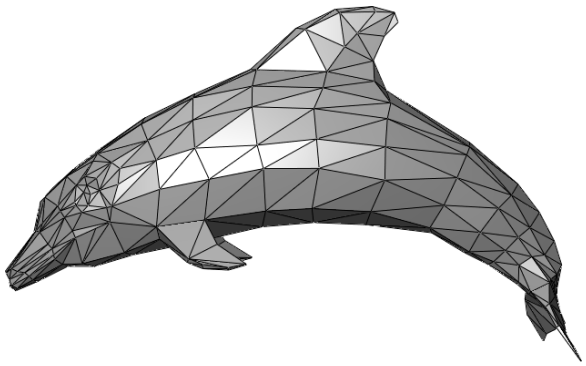

Figure 3: An example of a Polygon Mesh

Vertices are connected to form edges, which are connected to form faces. Faces are connected into polygons; polygons are connected into surfaces, and finally, the surfaces are connected into the object.[5]

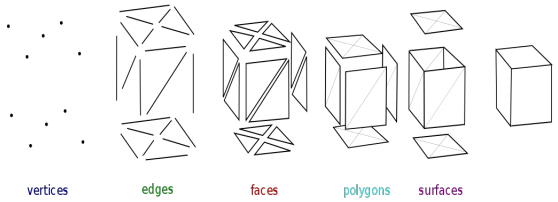

Figure 4: Mesh Overview

[4] https://commons.wikimedia.org/w/index.php?curid=10626667
[5] https://commons.wikimedia.org/w/index.php?curid=7021230

2.3.2 Face-Vertex Mesh

The most widely used mesh representation is the Face-vertex mesh. This representation is the input typically accepted by modern graphics hardware. The Face-Vertex Mesh stores model geometry in two parts. The first is a list of vertices. The second is a list of faces.[6]

- Vertex List: List of vertices (points in 3D space), with each vertex being defined by its x, y, and z coordinates. Each vertex also contains a list of faces that it belongs to. (This is redundant, but saves having to search the face list for faces that a specific vertex belongs to.)
- Face List: A list of faces (most commonly triangles), with each face being defined by the three vertices that make it up. It is of the form (v1, v2, v3) where v1, v2, and v3 are references to points in the vertex list (usually the index of the vertex in the vertex array).

  Since the connections between vertices do no change with normal geometric tranformations, most transformations only deal with the vertex list with the face list not being used until the final clipping and rasterization steps.

2.4 Camera Transformation

Once a world has been described with a World Coordinate System, and it has been populated by primitives, each with its own Object Coordination System, it is time to build the camera, or what the observer can see.

The camera transformation can be thought of as one more coordinate system: based upon the viewpoint of the observer, changing as the point of view changes. This coordinate system is defined by:[7]

- A scene of the world coordinates, including all primitives within those coordinates
- The camera position - $C = (x_c, y_c, z_c)$
- The point of the camera's focus - $A = (x_a, y_a, z_a)$
- The field of view angle - α
- The definition of the near and far planes, perpendicular to A with a distance of n and f from the camera

A pyramid is formed with C, A, and α which is then truncated by n and f into a frustum.

The transformation of the truncated viewing volume is given by:

[6] https://en.wikipedia.org/wiki/Polygon_mesh#Face-vertex_meshes

[7] http://idav.ucdavis.edu/education/GraphicsNotes/Viewing-Transformation/Viewing-Transformation.html#matrix

$$A_{\alpha,n,f} = \begin{pmatrix} \cot\frac{\alpha}{2} & 0 & 0 & 0 \\ 0 & \cot\frac{\alpha}{2} & 0 & 0 \\ 0 & 0 & \frac{f+n}{f-n} & -1 \\ 0 & 0 & \frac{2fn}{f-n} & 0 \end{pmatrix}$$

3 Geometry

3.1 Projection

Projection transforms the view volume into a unit cube with extreme points at (1, 1, 1) and (-1, -1, -1). This step is called projection even though it is a volume to volume transformation because the z-coordinate is saved for the rasterization step.[8]

3.1.1 Types of Projection

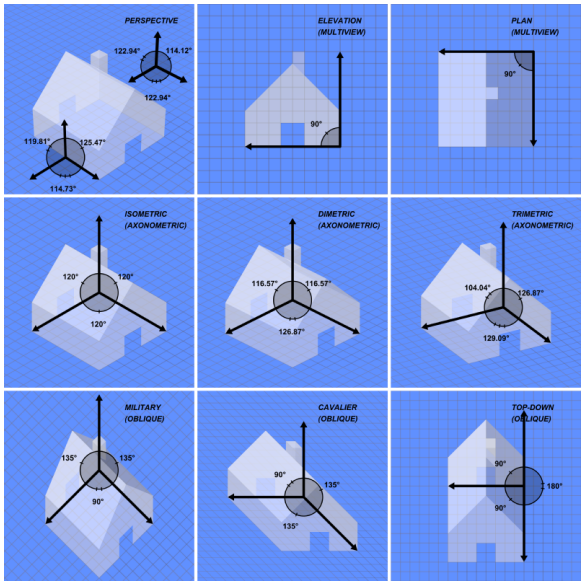

Figure 5: A comparison of projection types

3.1.2 Orthographic Projection

Orthographic projection is mostly used in engineering for multi-view drawings. It is to scale, and all parallel lines remain parallel with no forced perspective. An orthographic projection transformation can be represented as:[9]

[8] Real-Time Rendering, Akenine-Möller, T., Haines, E., Hoffman, N.
[9] Applied Geometry for Computer Graphics and CAD, Marsh, D.

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

For each homogeneous vector $v = (v_x, v_y, v_z, 1)$, the transformed vector would be

$$P_v = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \\ 1 \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \\ 0 \\ 1 \end{pmatrix}$$

3.1.3 Weak Perspective Projection

Weak perspective projection adds a scaling factor to orthographic projection. This scaling factor uses averages to approximate perspective and is used with simple models and small fields of view.[10]

$$P_x = \frac{X}{Z_{ave}}$$
$$P_y = \frac{Y}{Z_{ave}}$$

assuming focal length $f = 1$.

3.1.4 Perspective Projection

Perspective projection most closely replicates the human binocular vision where distant objects appear smaller. To describe the transformation, we must start with four variables:[11]

- $a_{x,y,z}$ - the 3D position of a point A that is to be projected
- $c_{x,y,z}$ - the 3D position of a point C representing the camera
- $\theta_{x,y,z}$ - the orientation of the camera
- $e_{x,y,z}$ - the display surface's position relative to the camera pinhole. (Although negative z values are more correct, the create an inverted image both horizontally and vertically, so most conventions use positive values.

To compute the 2D projection $(b_{x,y})$ of the point A $(a_{x,y,z})$ that is to be projected, a vector $d_{x,y,z}$ is first defined as the position of point A with respect to the camera transformation, with origin in C and rotated by θ:

[10] Applied Geometry for Computer Graphics and CAD, Marsh, D.
[11] Applied Geometry for Computer Graphics and CAD, Marsh, D.

$$\begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_x) & \sin(\theta_x) \\ 0 & -\sin(\theta_x) & \cos(\theta_x) \end{pmatrix} \begin{pmatrix} \cos(\theta_y) & 0 & -\sin(\theta_y) \\ 0 & 1 & 0 \\ \sin(\theta_y) & 0 & \cos(\theta_y) \end{pmatrix} \begin{pmatrix} \cos(\theta_z) & \sin(\theta_z) & 0 \\ -\sin(\theta_z) & \cos(\theta_z) & 0 \\ 0 & 0 & 1 \end{pmatrix} \left(\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} - \begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix} \right)$$

3.2 Clipping

Clipping saves time and energy by specifying only those objects within the view volume for rendering. The visual unit cube has been transformed into a truncated pyramid (frustum) and objects outside of the frustum are discarded by a method appropriately named frustrum culling. Objects that will be obscured by other primitives are also culled through backface culling. Objects inside the frustum are saved for rasterization.[12]

Objects that intersect the edges of the frustum must be clipped. Line clipping simply modifies the endpoints of lines to lie within the appropriate rectangle. Polygon clipping converts a polygon that intersects the frustum into one or more polygons the form the intersection of the original with the clip window. The Sutherland-Hodgman Polygon Clipping Algorithm is commonly used.[13]:

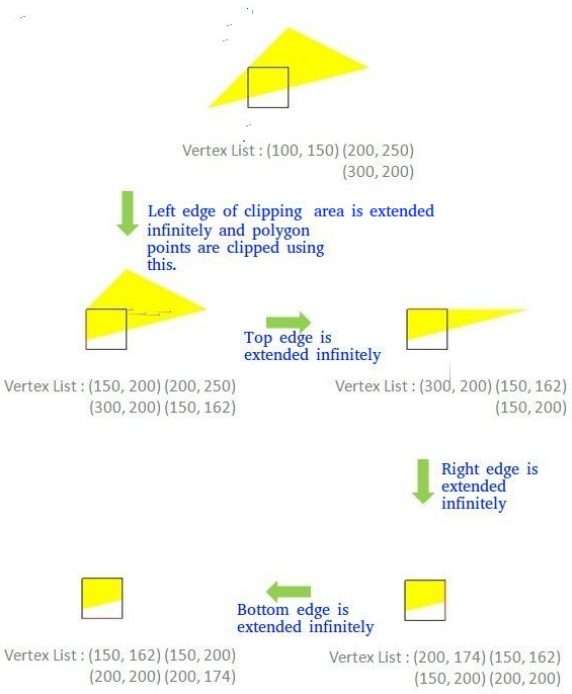

Figure 6: An example of the Sutherland-Hodgman Algorithm in use

[12] http://graphics.cs.cmu.edu/nsp/course/15-462/Spring04/slides/06-viewing.pdf

[13] https://www.geeksforgeeks.org/polygon-clipping-sutherland-hodgman-algorithm-please-change-bmp-images-jpeg-png/

3.3 Rasterization

Rasterization is the final step prior to display. The transformed vectors from the camera translation are converted into pixels. The rasterization step computes the color that each pixel on the screen should be given the relevant geometry, lighting, and shaders. Rasterization itself is a very complicated process, especially when taken into account things like lighting, textures, normal maps, and programmable shaders. Shading, lighting, z-buffering are all done at the rasterization step and help increase the illusion of 3D images on 2D screens.[14]

Finally, the image is displayed on the screen.

4 Conclusion

Modern graphics hardware and software have made much of the 3D graphics pipeline automatic. The math behind the transformations necessary to create worlds as different as Minecraft and Assassin's Creed are ultimately the same. The steps of the 3D graphics pipeline, Application, Geometry, and Rasterization, all make use of linear transformations represented by matrices.

[14] `https://en.wikipedia.org/wiki/Rasterisation`