Notes for ECE 30834 - Fundamentals of Computer Graphics

Zeke Ulrich

September 12, 2025

Contents

Course Description

Fundamental principles and techniques of computer graphics. The course covers the basics of going from a scene representation to a raster image using OpenGL. Specific topics include coordinate manipulations, perspective, basics of illumination and shading, color models, texture maps, clipping and basic raster algorithms, fundamentals of scene constructions.

Introduction

This class will be focused on *interactive* computer graphics over non-interactive graphics. In this class we will implement an interactive computer graphics engine in a basic programming language like C. A good understanding of professional and debugging and unit testing is required.

Let's look at a basic example of a GUI with some fundamental components.

Listing 1: GUI

```
#pragma once
#include gui.h
#include frambuffer.h

class Scene {
    GUI *gui;
    FrameBuffer *fb;
    Scene();
    void DBG();
    void NewButton(); // when this is clicked,
    // there is a callback to the
    // code that made the scene
}

extern Scene *scene;
```

Frame Buffer

The frame buffer stores the data that is going to be displayed on the screen. In the case of Listing 1 is a 1d array of unsigned integers.

Callback

Scene

Drawing

Animation

Linear Algebra

Points and Vectors

In three-dimensional space like ours, we require three numbers to specify a point. These numbers only make sense with reference to a coordinate system, typically an origin and set of axes like with Cartesian coordinates.

$$\mathbf{p} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{1}$$

$$d(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{2}$$

$$\mathbf{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = (v_x, v_y, v_z) \tag{3}$$

$$|\mathbf{v}| = \sqrt{v_x^2 + v_y^2 + v_z^2} \tag{4}$$

$$\hat{\mathbf{v}} = \frac{\mathbf{v}}{|\mathbf{v}|} = \frac{1}{\sqrt{v_x^2 + v_y^2 + v_z^2}} \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \tag{5}$$

Directions

A direction goes from one point to another, but its location in space doesn't matter. A direction starting at the origin going straight up is the same as a direction starting anywhere else going straight up. Its length doesn't matter either, so we generally keep it as length 1. We represent directions with 3D vectors, like $(1, 2, 3)$.

Dot Product

The dot product takes in two vectors and produces a number.

$$\mathbf{a} \cdot \mathbf{b} = a_x b_x + a_y b_y + a_z b_z = |\mathbf{a}||\mathbf{b}| \cos\theta \tag{6}$$

where θ is the angle between the vectors.

Cross Product

The cross product takes two vectors and returns a vector.

$$\mathbf{a} \times \mathbf{b} = \begin{pmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{pmatrix} \tag{7}$$

Matrix-vector Multiplication

Vectors and matrices can be multiplied.

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = \begin{pmatrix} m_{11}v_x + m_{12}v_y + m_{13}v_z \\ m_{21}v_x + m_{22}v_y + m_{23}v_z \\ m_{31}v_x + m_{32}v_y + m_{33}v_z \end{pmatrix} \tag{8}$$

Planes

A plane in three-dimensional space can be defined in several equivalent ways. One common form uses a point on the plane and a normal vector $\mathbf{n} = (a, b, c)$ perpendicular to the plane.

$$a(x - x_0) + b(y - y_0) + c(z - z_0) = 0 \tag{9}$$

This is the point-normal form of the plane, where (x_0, y_0, z_0) is a known point on the plane. It can also be written as:

$$ax + by + cz = d \tag{10}$$

where $d = ax_0 + by_0 + cz_0$.

If three non-collinear points P_1, P_2, and P_3 lie on a plane, we can compute two vectors from them:

$$\mathbf{v}_1 = \overrightarrow{P_1P_2}, \quad \mathbf{v}_2 = \overrightarrow{P_1P_3} \tag{11}$$

Then, the cross product of these vectors gives a normal vector to the plane:

$$\mathbf{n} = \mathbf{v}_1 \times \mathbf{v}_2 \tag{12}$$

With this normal and any of the three points, we can construct the plane equation.

Triangles

A triangle in 3D space is defined by three points: A, B, and C. These points form the triangle's vertices. The triangle can be described in terms of its edges, which are vectors:

$$\mathbf{AB} = \mathbf{B} - \mathbf{A}, \quad \mathbf{AC} = \mathbf{C} - \mathbf{A}, \quad \mathbf{BC} = \mathbf{C} - \mathbf{B} \tag{13}$$

The normal of the triangle's plane can be found by:

$$\mathbf{n} = \mathbf{AB} \times \mathbf{AC} \tag{14}$$

This normal is perpendicular to the surface of the triangle and is often used in graphics and geometry processing for lighting and orientation.

The area of the triangle is given by:

$$\text{Area} = \frac{1}{2}|\mathbf{AB} \times \mathbf{AC}| \tag{15}$$

You can also interpolate points inside the triangle using barycentric coordinates, which express any point P in the triangle as a weighted sum of the vertices:

$$P = \alpha A + \beta B + \gamma C, \quad \text{where } \alpha + \beta + \gamma = 1 \text{ and } \alpha, \beta, \gamma \geq 0 \tag{16}$$

Transformation

A transformation moves or reorients points, vectors, or shapes in space. The most common transformations are:

- Translation: Shifts a point by a vector.
- Scaling: Changes the size of an object.
- Rotation: Rotates an object around an axis.
- Reflection: Flips an object over a plane.
- Projection: Projects a point onto a plane or line.
- Translating Points and Vectors

To translate a point $\mathbf{p}$ by a vector $\mathbf{v}$:

$$\mathbf{p}' = \mathbf{p} + \mathbf{v} \tag{17}$$

Vectors are not affected by translation—they describe direction and magnitude, not position.

To translate a triangle, add the translation vector $\mathbf{t}$ to each vertex:

$$A' = A + \mathbf{t}, \quad B' = B + \mathbf{t}, \quad C' = C + \mathbf{t} \tag{18}$$

The shape and size of the triangle remain unchanged.

Linear transformations like rotation, scaling, and shearing are usually done via matrix-vector multiplication. A 3x3 matrix transforms vectors or points (relative to the origin):

$$\mathbf{v}' = M\mathbf{v} \tag{19}$$

To include translation with linear transformations, we use homogeneous coordinates and 4x4 matrices, which is common in computer graphics.

Rotations

Rotations can be encoded with matrices. In three-dimensional space, rotation is a linear transformation that turns vectors around an axis while preserving their lengths and angles between them.

Given an angle θ, rotating a vector by θ is accomplished with a rotation matrix. The exact form of the matrix depends on the axis of rotation.

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix} \tag{20}$$

$$R_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \tag{21}$$

$$R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{22}$$

To rotate a vector $\mathbf{v}$ about an axis, we multiply it by the corresponding rotation matrix:

$$\mathbf{v}' = R\mathbf{v} \tag{23}$$

For example, rotating $\mathbf{v}$ about the z-axis by θ radians:

$$\mathbf{v}' = R_z(\theta)\mathbf{v} \tag{24}$$

To rotate around an arbitrary unit axis vector $\hat{\mathbf{u}} = (u_x, u_y, u_z)$ by an angle θ, we can use Rodrigues' rotation formula or construct a 3x3 matrix:

$$R(\hat{\mathbf{u}}, \theta) = \cos\theta, I + (1 - \cos\theta), \hat{\mathbf{u}}\hat{\mathbf{u}}^\top + \sin\theta, [\hat{\mathbf{u}}]_\times \tag{25}$$

Where I is the identity matrix, $\hat{\mathbf{u}}\hat{\mathbf{u}}^\top$ is the outer product, and $[\hat{\mathbf{u}}]_\times$ is the skew-symmetric matrix.

$$[\hat{\mathbf{u}}]_\times = \begin{pmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{pmatrix} \tag{26}$$

This general rotation formula allows rotation around any axis, not just the coordinate axes.

Changing Coordinate Systems

Given a point P, a new coordinate system defined by an origin O' and basis vectors $\vec{x}, \vec{y}, \vec{z}$, how do we find the coordinates of P in this new system?

First, we express the vector from the new origin to P:

$$\mathbf{v} = \mathbf{P} - \mathbf{O'} \tag{27}$$

This gives the position of P relative to the new origin.

Next, we project $\mathbf{v}$ onto the new basis vectors. Assuming the new basis is orthonormal (i.e. all vectors are unit length and perpendicular to each other), we compute:

$$P'_x = \mathbf{v} \cdot \mathbf{x'} \quad P'_y = \mathbf{v} \cdot \mathbf{y'} \quad P'_z = \mathbf{v} \cdot \mathbf{z'} \tag{28}$$

So P in the new coordinate system is:

$$\mathbf{P'} = \begin{pmatrix} P'_x \\ P'_y \\ P'_z \end{pmatrix} \tag{29}$$

Matrix Form (Basis Change Matrix)

You can also express this operation as a matrix multiplication. Let R be the rotation matrix whose columns are the new basis vectors:

$$R = \begin{pmatrix} | & | & | \\ \mathbf{x'} & \mathbf{y'} & \mathbf{z'} \\ | & | & | \end{pmatrix} \tag{30}$$

Then the change of basis is:

$$\mathbf{P'} = R^\top (\mathbf{P} - \mathbf{O'}) \tag{31}$$

Cameras

In graphics, a camera projects a 3D scene onto a 2D plane which can be displayed on a screen. If we model the camera as a point, then the problem becomes a linear algebra issue of projecting a set of points onto a plane determined by the camera. We must display only surfaces not obscured by other surfaces, not accounting for things like transparency.

The camera is a grid of pixels and an eye. The eye shoots out rays to the scene and where those rays intersect the plane, draws a point.

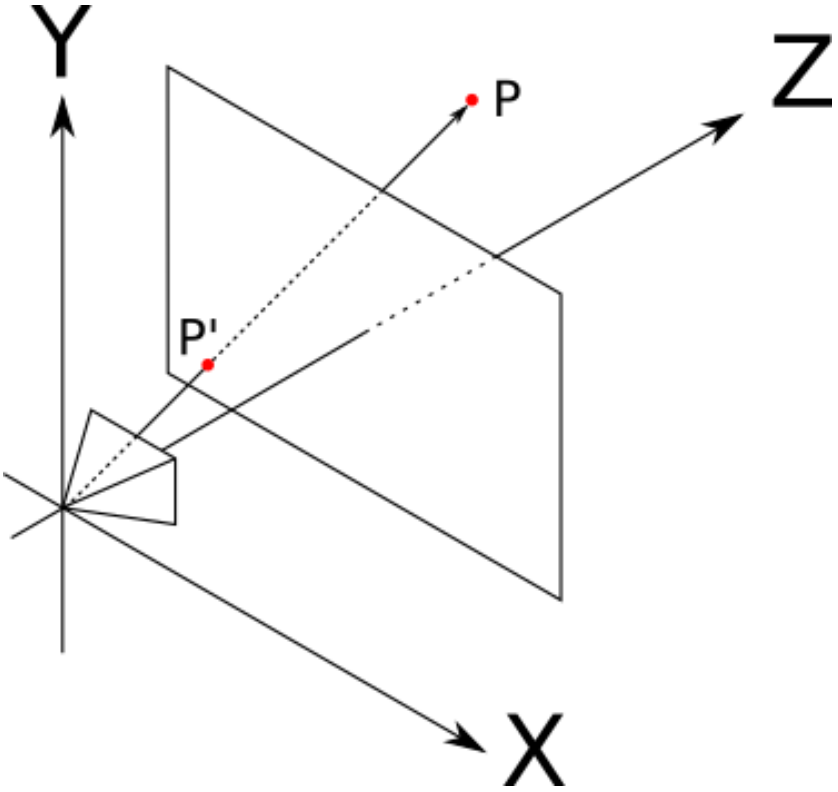

Figure 1: Camera

A *pose* consists of three translations and three rotations. Setting each results in a unique camera pose in 3D space.

The issue of projecting a point onto a plane involves linear algebra. There are known formulas for it, which are not listed here.

The probability that a ray exactly intersects the center of a pixel is zero. This means we have to calculate the pixel center closest to the intersection and use that instead.

The human eye has the wonderful property that it sees things far away in a panorama, and things that are in close in intricate detail.

Translations of the camera must be with respect to its current orien-

There are several ways to approximate the way the human eye sees, the method used in this class is called the planar pinhole method. There are many, many other camera models, for instance thin lens camera, orthographic camera, spherical camera, fisheye camera.

The probability that a ray exactly intersects the center of a pixel is not zero, since floating point numbers have a finite range of values they can take, but practically it is zero.

tation. That is, the camera must move left or right relative to what it sees, which is more difficult than just changing the x coordinate.

Drawing

Drawing more complex shapes just requires connecting vectors together and displaying them on screen. We can translate them, rotate vectors, and shift the camera.

Triangles

In computer graphics, everything is made of triangles. A *shared-vertex triangle mesh* is a bunch of triangles who can have the same vertices that forms some shape. A triangle is fully specified with a triple of unsigned ints that specify which indices in the array of vertices belong to that triangle. For instance, with a cube made of 8 vectors, each triangle needs only specify which three of the vectors that form the vertices it has.

Once we have the ability to draw triangles, it would be useful to be able to draw points and wire frames.