Tecnológico de Monterrey

Advanced Programming

Prof. Gilberto Echeverría

Final Project: 3D Graphics

Marcelo Schonbrunn Bonnin A01207573

The following document will detail the process of implementation for a basic 3D rendering Engine.

DATA TYPES

There are 4 main Data types required when working with 3D space, each expands on the previous, meaning that they are intertwined and crucial to each other. The data types will be described using the following format:

- **DATA TYPE:** DESCRIPTION/EXPLANATION
    - Primitive 1
        - Primitive Explanation
    - Primitive 2
        - Primitive Explanation
    - etc
- **Vec3D**: Often referred to as vector 3D or 3D Vector; the vec3d data type can be thought of as a point in 3D space, it is a necessity of all 3D rendering systems. Although most modern graphics systems use quaternions.
    - Float x
        - X coordinate in 3D space
    - Float y
        - Y coordinate in 3D space
    - Float z
        - Z coordinate in 3D space
    - Float w
        - Used to keep track of perspective, can be used to scale the previous 3 points.
- **Triangle:** A triangle is the most basic geometric shape since it can be made with the fewest number of points(3). A triangle is a collection of 3 separate vec3d elements. It should be noted that the set of points that can make up a triangle != the set of all 3d points
    - Vec3d p[3]
        - Array that holds 3 vec3d elements

- **Mesh:** A mesh is a dynamic collection of triangles, theoretically it can hold an infinite number of triangles but practically it is limited by system memory. The mesh can be thought of as the wireframe of a 3D object.
    - vector<triangle> triangles
        - Vector object that holds the triangles that make up an object
- **Mat4d:** A four dimensional matrix. matrices of this kind are mostly used as helpers to properly display an object. Their functionality includes, but is not limited to: Rotating objects, Projecting objects, translating objects, etc. Important matrices are listed below, see **IMPORTANT MATRICES** for more details.
    - Float m[4][4]

With these four different data types a 3D engine is capable of displaying most 3D objects.

IMPORTANT MATRICES

The following section will detail the most noteworthy matrices used in the 3D engine. Do note that all matrices in this section belong to the Mat4d data type unless explicitly stated.

- **projMat:** The projection matrix[1] allows a vector space to be 'projected' into a subspace. In this case it is used to project a 3D vector into 2D space[2]. This is important because most monitors are only capable of displaying 2D objects, therefore, all 3D objects must be converted into 2D objects by considering their z position in space as well as the physical attributes of the medium they are being presented on, in this case, that mostly refers to monitor resolution.

$$S = \frac{1}{\tan(\frac{fov}{2} * \frac{\pi}{180})}$$

[1]https://mathworld.wolfram.com/ProjectionMatrix.html#:~:text=A%20projection%20matrix%20is%20an,projection%20matrix%20is%20orthogonal%20iff

[2]https://www.scratchapixel.com/lessons/3d-basic-rendering/perspective-and-orthographic-projection-matrix/building-basic-perspective-projection-matrix

$$\frac{\frac{-(z'=z=-n)*f-f*n}{(f-n)}}{(w'=-1*z=n)} = \frac{\frac{n*f-f*n}{(f-n)}}{(w'=-1*z=n)} = 0$$

$$\frac{\frac{-(z'=z=-f)*f-f*n}{(f-n)}}{(w'=-1*z=f)} = \frac{\frac{f*f-f*n}{(f-n)}}{(w'=-1*z=f)} =$$

$$\frac{\frac{f*(f-n)}{(f-n)}}{(w'=-1*z=f)} = \frac{f}{f} = 1$$

$$\begin{bmatrix} S & 0 & 0 & 0 \\ 0 & S & 0 & 0 \\ 0 & 0 & -\frac{f}{(f-n)} & -1 \\ 0 & 0 & -\frac{f*n}{(f-n)} & 0 \end{bmatrix}$$

- **Transformation Matrix:** A transformation matrix[3] allows for the modification or 'transformation' of another matrix or vector. In our case, we simply use transformation matrices to rotate our objects in 3D space. There are 2 transformation matrices used in this particular program: matRotX and matRotZ. They allow for individual rotation in the x and z axis.

matRotZ

$$\begin{bmatrix} cosf(\Theta) & sinf(\Theta) & 0 & 0 \\ -sinf(\Theta) & cosf(\Theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[3] https://en.wikipedia.org/wiki/Transformation_matrix

matRotX

$$\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & cosf(\Theta) & sinf(\Theta) & 0\\ 0 & -sinf(\Theta) & cosf(\Theta) & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

- **Identity Matrix:** An identity[4] matrix is the most basic diagonal matrix[5] where all non zero values are 1. In essence, it is a square matrix that has a diagonal sequence of 1s that goes from [0][0] to [1][1] and every other value is 0. Identity matrices are multiplied with other matrices throughout the program to extract x,y,z,w (vec3d) values.

4x4 Identity Matrix

$$\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

NORMALIZATION:

Normalizing[6] refers to breaking down a vector into its unitary components so that you end up with a vector that has a magnitude <= 1.

$$||\vec{v}|| = \sqrt{v_x * v_x + v_y * v_y}$$

(for vectors with more components, just add the square component into the root)

[4] https://mathworld.wolfram.com/IdentityMatrix.html
[5] https://mathworld.wolfram.com/DiagonalMatrix.html
[6] http://www.fundza.com/vectors/normalize/

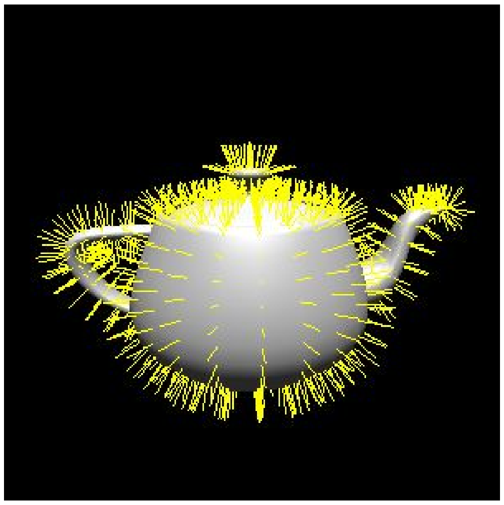
Once normalized, the resulting unitary vector can be compared to other unitary vectors using the dot product[7] to obtain a value between -1 and 1. Uses for this value are covered below in the **RENDERING** section.

$$\begin{bmatrix} A_x & A_y & A_z \end{bmatrix} \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} = A_x B_x + A_y B_y + A_z B_z = \vec{A} \cdot \vec{B}$$

(Dot Product Function)

NORMALS:

A normal[8] is a unitary vector, found through normalization which describes the orientation a triangle in a mesh has. Because every triangle has 3 vec3d elements, 3 different normals can be calculated for it. In the name of optimization however, the three points that make up the triangle can be averaged, to find the center of the triangle. This point is then used to calculate the average normal for any given triangle.



[7] https://betterexplained.com/articles/vector-calculus-understanding-the-dot-product/
[8] https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/shading-normals

RENDERING:

In this case, rendering essentially refers to drawing triangles on screen. Though it may sound simple, it is important to consider which triangles should be rendered. This is because objects are solid and forward facing triangles should block out triangles in back faces. In order to figure out which triangles to render and which to ignore, a dot product, involving the normal of a triangle and the normalized camera position can be calculated. The result will be in the range -1 to 1 (as stated previously) and based on our particular implementation we either can choose to draw triangles with a value > 0 or triangles with a value < 0.

LIGHTING:

Lighting[9] refers to the intensity with which an object is illuminated. Each drawn triangle may have a different amount of illumination. There are many different types of lights in 3D graphics but for the sake of simplicity a directional light[10] (which can't actually be found in the real world) will be used. In the most simple of terms, a directional light is a point in 3D space where light is emitted in a single direction. In order to calculate the lighting of our objects we are employing a very similar technique to the one described above in the **RENDERING** section. If we consider our light to simply be a point in 3D space and its intensity to be represented on a scale from 0.0 to 1.0, we can calculate the dot product of the normalized light vector and the normal of a particular triangle. If the result is in the range $0 < x < 1$ then it is a direct representation of the brightness of that face and if its $x <= 0$ we know that the triangle is receiving no light.

[9] https://www.cs.uic.edu/~jbell/CourseNotes/ComputerGraphics/LightingAndShading.html
[10] https://learn.foundry.com/modo/902/content/help/pages/shading_lighting/lights/directional_light.html