EXPLANATION OF THE ALGORITHMS FOR DISPLAYING 3D FIGURES ON THE COMPUTER SCREEN

Alexander Zhyrytovskyi
Institute of Computer Technologies, Graduate School
Open International University of Human Development “Ukraine”
23 Lvivs’ka str., Kyiv, Ukraine. 04071
i.am.zhirik@gmail.com

Abstract

Not long time ago people can only dream about the thing that they can see 3-dimensional world inside of the screen of the computer. Computers at that time can display only 256 colors on their screens, they can work only with integer numbers, and their computation speed was not fast enough. This dream comes true with the help of Id Software company. This company started to build its games by using of simplified 3D graphics. All these simplified 3D graphics were calculated by the software 3D rendering algorithms which were not published by these company. At that time there was a global goal to speedup calculations for 3D graphics and to make it look more realistic. The solution for this comes from hardware companies. Hardware companies started to work on hardware 3D accelerators, which can render 3D graphics much faster than software algorithms. These 3D accelerators are fully patented, so that no one knows how they are implemented inside. At that time there were proposed two 3D rendering interfaces: Direct3D and OpenGL. And manufacturers of graphic cards started supporting both of these interfaces. Lots of game development companies started to use these interfaces. And also Id Software company started to redesign its game engine to support these interfaces to take place in game market. From that time, everybody started to use 3D accelerators with built-in 3D rendering features and as a result everybody forget about software 3D rendering algorithms.

Nowadays the situation come in the following way, that most of the algorithms which are used in 3D games are not published on paper, and all the 3D accelerator internal algorithms are patented. All modern 3D graphics are fully conserved by the video cards, so that 3D rendering could not be developed in other way.

The goal of this article is explanation of how 3D graphics can be displayed on the screen of the computer

Keywords: 3D, graphics, rendering, mesh, z-buffer, texture, projection, perspective, triangulation, triangle drawing.

DOI: 10.21303/2461-4262.2018.00680

© Alexander Zhyrytovskyi
1. Introduction

3D graphics nowadays uses in games, movies, cartoons and for engineering process. Most of things such as cars, planes, houses and portable devices before being implemented should have 3D model of itself and also 3D models of its internal parts. Therefore, 3D graphics will be in demand for the near future.

Today probably every person seen the 3-dimensional figures inside the computer, but nobody can figure out what formulas and what operations were performed to build these figures. During this article, it is possible to see all the basic steps that need to be done to build 3D figures on the screen of the computer.

2. 3D rotation formulas

The most fundamental part in 3D graphics is 3-dimensional rotation formulas, which looks the following:

\[ x' = x \cos a - y \sin a; \]
\[ y' = (x \sin a + y \cos a) \cos b - z \sin b; \]
\[ z' = (x \sin a + y \cos a) \sin b + z \cos b, \]

here \(x, y, z\) – coordinates of the initial source point; \(x', y', z'\) – coordinates of the rotated final point; \(a, b\) – horizontal and vertical angles of the rotation.

2.1. Flat projection

To project 3D world point coordinates \(x', y', z'\) into the screen point coordinates \(x'', y''\), it is possible to use the following formula:

\[ x'' = x'; \]
\[ y'' = y'. \]

Here for projection just exclude \(z'\) variable. As an example on Fig. 1, it is possible to see the figure that was previously rotated by 3D rotation formulas, and now it is projected down to the screen using flat projection formulas.

3. Projection with perspective

In real world, you can look at the road and you can see that this road becomes smaller as far it is located from us (Fig. 2).
In some distance from us, the object becomes really small so that we stop seeing it. The distance from the screen to the center of perspective comes here as value of perspective (Fig. 3).

Such projection is described using following formulas:

\[ x'' = x' \frac{d}{d+z}; \]
\[ y'' = y' \frac{d}{d+z}, \]

here \( x', y', z' \) – coordinates of the initial source point; \( x'', y'' \) – coordinates of the final rotated point; \( d \) – value of perspective.

4. **Drawing 3D figure using dots**

The simplest 3D figures can be built using dot-by-dot projection to the screen. The example of such figure is displayed on Fig. 4.

4.1. **Drawing 3D figure using mesh**

Other approach is to use lines instead of dots. In this case, all the projected points we can connect into mesh of lines (Fig. 5).
4.2. Triangulation
To make the figure solid it is possible to build it using mesh of solid triangles (Fig. 6).

4.3. Triangle drawing algorithm
To draw the triangle itself, there is triangle drawing algorithm. On Fig. 7, it is possible to see how triangle drawing algorithm works.

Algorithm:
1. Sort triangle vertices so that: $Y_1 \leq Y_2 \leq Y_3$;
2. Create two points $P_1$ and $P_2$ (Fig. 7). At the beginning both of these points will have coordinates $(X, Y)$;
3. Calculate horizontal increments $d_1$ and $d_2$, and also number of iterations $N$. It is possible to do this by the following formulas:

$$d_1 = \frac{X_2 - X_1}{Y_2 - Y_1};$$
\[ \begin{align*}
    d_2 &= \frac{X_2 - X_1}{Y_2 - Y_1}; \\
    N &= Y_2 - Y_1.
\end{align*} \]

1. Repeat it N times: move the points \( P_1 \) and \( P_2 \) by the one step to the bottom. Also, it is necessary to move horizontally \( P_1 \) by the increment \( d_1 \). And also to move horizontally \( P_2 \) by the increment \( d_2 \). And fill empty horizontal place between \( P_1 \) and \( P_2 \) using dots;

2. Recalculate first increment \( d_1 \) and number of iterations \( N \):
\[ \begin{align*}
    d_1 &= \frac{X_2 - X_1}{Y_2 - Y_1}; \\
    N &= Y_3 - Y_2.
\end{align*} \]

3. Repeat it N times: move the points \( P_1 \) and \( P_2 \) by the one step to the bottom. Also it is necessary to move horizontally \( P_1 \) by the increment \( d_1 \). And also to move horizontally \( P_2 \) by the increment \( d_2 \). And fill empty horizontal place between \( P_1 \) and \( P_2 \) using dots.

4. **Z-buffer**

   When drawing solid triangle mesh on the screen if these triangles are collapsed, it is possible to see visual ordering problem (Fig. 8).

   ![Fig. 8. Triangle order displaying problem](image)

   Basically this problem can happen if on the same place we need to draw 2 points with different colors and different z-distances. And the problem is how to keep the point with minimal z-distance. To solve this problem, it is necessary to use extra image which will contain z-distances for all the points that are displayed on the screen, this image is called z-buffer, it is possible to see it on Fig. 9.

   ![Fig. 9. Normal image and z-buffer image](image)

   Let’s suppose that at the beginning z-buffer filled with extremely big numbers. When we want to put a point on the screen, we can do this only if its z-distance is smaller than z-distance of corresponding point of z-buffer. After when we put the point on the screen we also need to put its z-distance to z-buffer.
5. Filling the triangle with 3 color gradient

Triangle has 3 vertices. In this approach, each triangle vertex has its coordinates and color. To find the color of the current point inside triangle it is necessary to find the distances to each of the vertices (Fig. 10).

![Fig. 10. Filling triangle with 3 color gradient](image)

Let’s find these distances by the following formulas:

\[
L_1 = \sqrt{(X - X_1)^2 + (Y - Y_1)^2};
\]

\[
L_2 = \sqrt{(X - X_2)^2 + (Y - Y_2)^2};
\]

\[
L_3 = \sqrt{(X - X_3)^2 + (Y - Y_3)^2},
\]

here \(X, Y\) – coordinates of current point inside triangle; \(X_1, Y_1\) – coordinates of first vertex of the triangle; \(X_2, Y_2\) – coordinates of second vertex of the triangle; \(X_3, Y_3\) – coordinates of third vertex of the triangle; \(L_1, L_2, L_3\) – distances from current point to each of vertices.

Using these distances, it is possible to calculate the color of current point by the following formulas:

\[
R = \frac{R_1 \cdot L_1 + R_2 \cdot L_2 + R_3 \cdot L_3}{L_1 + L_2 + L_3};
\]

\[
G = \frac{G_1 \cdot L_1 + G_2 \cdot L_2 + G_3 \cdot L_3}{L_1 + L_2 + L_3};
\]

\[
B = \frac{B_1 \cdot L_1 + B_2 \cdot L_2 + B_3 \cdot L_3}{L_1 + L_2 + L_3};
\]

here \(R_1, G_1, B_1\) – red, green and blue components of the vertex 1 color; \(R_2, G_2, B_2\) – red, green and blue components of the vertex 2 color; \(R_3, G_3, B_3\) – red, green and blue components of the vertex 3 color; \(R, G, B\) – red, green and blue components of the destination color for the current point.

5. 1. Texture overlay on 3D figure

Texture overlaying on triangle is very similar to filling it with 3 color gradient, the only difference is that instead color component variables it is necessary to use texture coordinate variables (Fig. 11).

To draw 3D figure with texture overlay, first it is necessary to take some texture image and divide it into triangles. This dividing means that for each triangle we need to calculate its coordinates on texture (Fig. 12).
Fig. 11. Drawing texture on the triangle

After that, it is possible to change coordinates of the mesh with triangles, and produce any 3D figure with texture (Fig. 13).

Fig. 12. Overlaying the texture over the mesh of triangles

Fig. 13. Wrapping mesh of triangles into different shapes

6. Conclusion

Theoretical algorithms for building 3D figures, that were provided in this article, were implemented and tested. It is possible to see their implementation in software program, which can be viewed by the following link: http://alex.zhyrytovskiy.x10.name/progs/toys/texture3D.zip

As the result using these algorithms, it is possible to build hardware independent 3D graphics which can be calculated in real time without usage of any graphic card interfaces such as DirectX or OpenGL.

Also nowadays, there looks to be no literature that describing 3D rendering process, and possibly this article can open eyes of the reader on 3D rendering process and will help him to understand how it performs.
References
[16] Z-buffer channel. Available at: http://support.nextlimit.com/display/maxwell4/Z-buffer+ channel