Parallel 3D Engine

Ivan Garcia Alsina
Advisor: Dr. Jaime Seguel

Electrical and Computer Engineering Department
University of Puerto Rico, Mayaguez Campus
Mayagüez, Puerto Rico 00680-5582
ivangarcia44@hotmail.com

Abstract

This paper is about the design of a parallel 3D engine. This engine has all the essential features required for animations, simulations or video games. The engine's algorithm was designed in C. After the engine was complete, it was translated from C to Cilk and tested in a shared memory computer with 4 processors.

1. Introduction

A 3D engine is a software component that can be used to make animations, simulations or video games. One of the most popular 3D engines among professional game developers is DirectX. DirectX can be used as an ActiveX for Visual C++, Visual Java and any other language that support ActiveX. One of the most important functions of DirectX is drawing 3D polygons in space. It moves and rotates the observer and polygons in space. The engine has polygon rendering, bitmaps, light effects, sound, collision detection and other features.

The engine in this work is developed in DOS instead of Windows. The engine has the basic features of rotating and moving (total of 6 degrees of freedom) the observer and objects, and clipping. The 3D engine algorithm was designed for implementation in parallel.
All the operations on the polygons can be performed separately, except for a sorting algorithm discussed later. Thus, N different polygons can be distributed in N parallel processes. The parallel engine was developed in Cilk and MPI. The runtime tests showed that the Cilk engine was superior to the MPI engine. Since the computer used had 4 processors the MPI engine could only divide polygon operation in four parallel processes. In contrast, the Cilk engine could divide each transformation in N processes, which makes the engine faster. The MPI engine is also slower because large data sets are to be transferred between processes. The Cilk engine instead, does not have to transfer data because it runs on a shared memory environment.

2. Engine Algorithm

The polygon data is read from the hard disk only once. Then it is transformed dynamically, displayed on the screen. When the user exits the program, the polygon data is deleted.

Following is a list of the processes performed by the engine.

1) Create 3D points.

2) Transform 3D polygons using observer and object frames.
3) Clip the 3D polygons.
4) Create 2D points.
5) Transform each 3D point into a 2D point.
6) Get polygons view order.
7) Put polygons in view order.
8) Clip the 2D polygons.

Four of these steps can be executed in parallel (2, 3, 5 and 8). Each process is explained in more detail in the following sections.

3. Polygon Linked List

The polygon data is stored in a linked list. The following structure composes each node of the linked list.

```
struct polygon {
   int N, C, Cb;
   FRAME *pos;
   P3D *Pt;
   P3D *Pt3d;
   P2D *Pt2d;
   int rank;
   struct polygon *next;
   struct polygon *prev;
 };
```

The rank is used to store the order in which the objects are seen by the observer. The linked list Pt stores the values of the original points read from the hard disk. The list Pt3d stores the points, which have been operated in three dimensions, and the list Pt2d stores the points in two dimensions. The variable N is the original number of vertices in the polygon. C is the surface color and Cb the border color. The next and prev variables point to the next and previous nodes in the linked list respectively.

4. Point Transformation Using Reference Frames

The observer and the three dimensional object are represented by reference frames. A reference frame is composed of an origin with three coordinates and a set of three vectors that represent orientation.

The sum of variables of a reference frame is twelve, three for the origin and three for each vector. Each of those vectors has three components: x, y and z. Those components are parallel to the fixed reference frame of the 3D engine.

Rotations on a plane normal to the vector Vz are performed by rotating the Vx and Vy vectors. The same logic is used to rotate on the planes that are normal to vectors Vx and Vy. To move the origin of a frame, the magnitude of displacement is multiplied by the frame's vector parallel to the direction of the displacement.

The polygon data loaded from the hard disk are relative to its object reference frame. To obtain the coordinates relative to the observer: we transform first, the coordinates related to the object to coordinates related to the fixed reference frame. Then, we transform these coordinates in order to relate them to the observer.

5. 3D to 2D Point Conversion and 3D Clipping

The 3D engine makes these processes for all points in the polygon linked lists. The

processes take each polygon Pt list as parameter and the result is stored in the Pt3d list. After doing the reference frame transformation (explained in the previous section), the engine makes the 3D clip in each polygon Pt3d list. The clip 3D function eliminates all polygons behind the observer, and if they are in between, it cuts them. To cut a polygon, the engine eliminates the points behind the observer and adds the necessary points in the list to preserve the shape of the polygon.

After the 3D clipping is complete the engine creates the list Pt2d parallel to the Pt3d list. Points in the Pt3d and Pt2d list are related as follows:

(X3d, Y3d, Z3d) = Point from Pt3d list.
(X2d, Y2d) = Corresponding point from Pt2d list.

X2d = (268.0*X3d) / (Z3d+1.0);
Y2d = (201.0*Y3d) / (Z3d+1.0);

The constants 268.0 and 201.0 are obtained by calibration and are used to make the engine look more realistic.

6. Polygon Sorting

Asume that we have the polygons of a desk, a computer, a T.V. set and a wall stored in the engine. Without sorting, the engine could draw the wall first and therefore, we will not be able to see the other objects. In order to avoid this problem we use a polygon sorting algorithm. The algorithm puts the polygons in the inverse order in which the observer sees them. For example, the order for the four objects above could be wall, T.V., computer and desk.

A rank is assigned to each polygon depending on the order in which the observer sees them. When the sorting is completed the engine swaps the polygons in the linked list using the rank variables.

The sorting algorithm is equal to a linear sorting algorithm. The only difference is in the function used to compare elements. This function returns a zero if the polygons compared do not overlap, a one if the first polygon is behind the second and a minus one if the first polygon is in front of the second. If the returned value of the comparison function is 1 then a swapping is performed.

To determine if one object is behind the other the function executes the following process. First, it searches for an intersection of the polygon's 2D projections. If no intersection is found the function returns zero indicating that they do not overlap. If the intersection is found, its 3D equivalent is calculated for both polygons. Then the distance from the 3D points of each polygon and the observer is calculated. If the distance of the first polygon is greater than the second polygon's distance then, the first polygon is behind, and the function returns one, otherwise it returns minus one.

7. 2D Clipping

Before drawing the polygon list, the 2D points must be adjusted to the screen. If a polygon goes out of the screen limits, it is deleted. If the polygon is between the valid range and the limits, the polygon is clipped. After adjusting the coordinates of the screen limits we invert the Y

screen coordinates and displace them (this depends on the screen mode used).

8. Parallel 3D Engine

To test if the serial engine worked properly, the data of a labyrinth was loaded and the observer was displaced and rotated to all critical situations. The original 3D engine was developed in Borland C for DOS. To develop the parallel engine, the serial engine had to be translated from a DOS C compiler to a Sun C compiler. Since this compiler did not had graphics, the data generated from the DOS engine was used to validate the parallel engine.

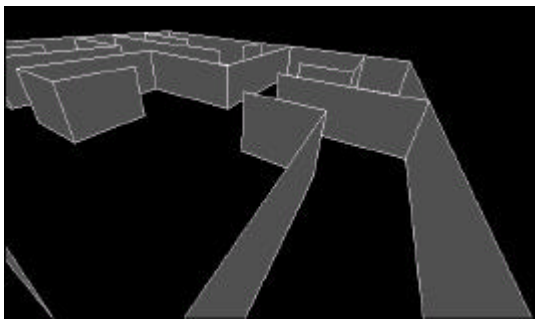

Figure 1. DOS 3D Engine Snap Shot

Not all the engine's process could be executed in parallel. Because of the large amount of communications involved, the sorting algorithm was implemented serially. The allocation of memory was also made serially.

Since the remaining processes are independent, they could be made in parallel. These processes are the reference frame transformation, 3D and 2D clipping and 3D to 2D point conversion. For example, if we are going to clip the polygons in three dimensions, we execute the clipping of each polygon in parallel. If we have to clip N polygons, the clipping procedure runs N times faster. The Cilk commands spawn and sync was used to achieve this goal.

References

[1] Diesel, P.J. and Deitel, H.M., "C How to Program", Prentice Hall 1994: 466-519

[2] Hearn, Donald and Baker, M. Pauline, "Computer Graphics", Prentice Hall, 1994: 296-468

[3] Pytel, Andrew and Kiusalaas, Jaan, "Engineering Mechanics Statics and Dynamics", Harper Collins College Publishers 1994: 467-479

[4] Thomas, George B. and Finney, Ross L., "Calculus and Analytic Geometry, 9th edition", Addison Wesley 1996: 787-829