Theory of Computer Graphics

Galdan MOULINNEUF

2927686

Question 1:

« Rendering pipeline » is a sequence of steps to create a 2D representation of a 3D scene.

The first step is called “Vertex Processing”, it takes “vertices” as input (points in the space that have coordinates (x, y, z). The “Vertex processing” is divided in two steps: “Vertex Shader” which will transform the vertices into “vertexes” by computing where the camera is looking, then, the “Primitive Assembly” will link the vertexes together if possible.

The second step is called “Rasterization”. Its work is pretty much like the “Primitive Assembly” but it will transform the primitives into fragments. Those fragments will get attributes like their pixel position, interpolated colour, depth or texture coordinates.

The last step is the “Fragment Processing” which consists of computing for each pixel, the colour it must be thanks to the fragments.

After all those steps, the render is ready to be displayed.

Sources: lectures, Youtube video

Question 2:

A frame buffer is an array used to store the image that will to be displayed (so the output of the rendering pipeline). Its size depends of the resolution of the image and the number of colour that a pixel can take, the formula that calculate the size is:

$$size = width * length * nb\_colours\_in\_bits$$

A full colour pixel is a 24bits long pixel, which means that in the frame buffer, a pixel will be represented like that in hexadecimal: 0xFFFFFF for a black one.

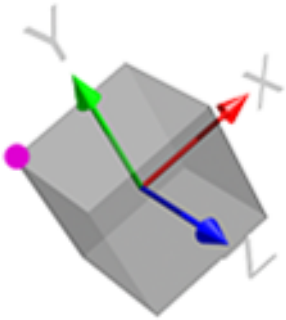
A true colour pixel is 32bits long. The extra byte is used for the alpha channel, most commonly known as opacity. Here is a representation in hexadecimal: 0xFFFFFF7F for a black pixel with half opacity.

Sources: lectures, “What is True Color”

Question 3:

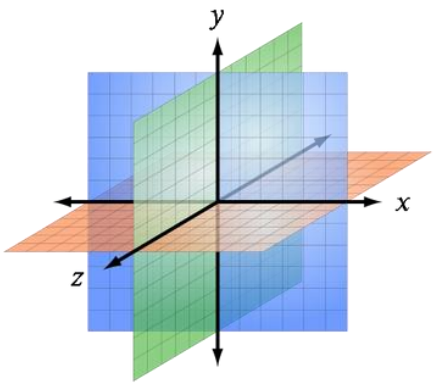
- Object Coordinate System:

  To create a 3D model, you must pick a point to be the origin of that model and an orientation the set the axes. It's proper to each model, for example, if the model rotates, it uses his own coordinates.



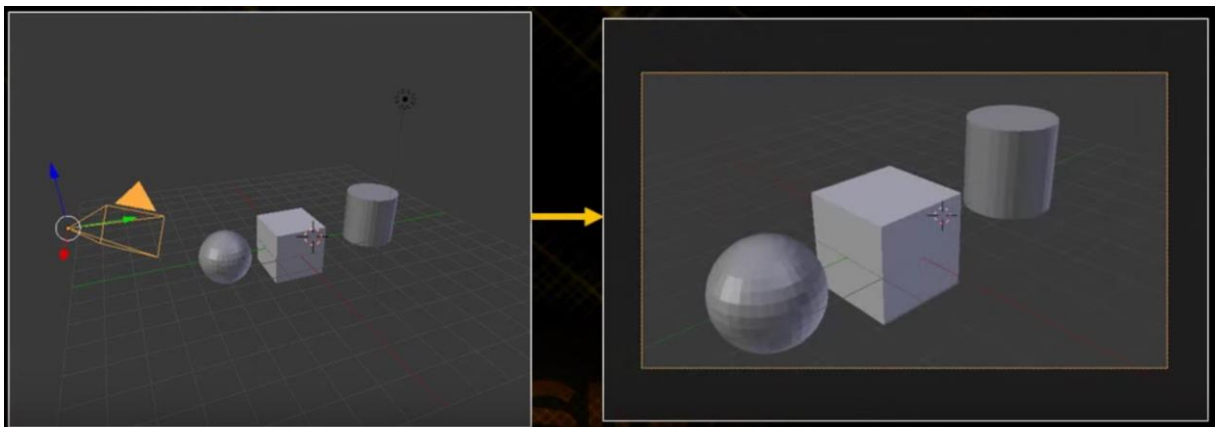
- World Coordinate System:

  This is the base reference system for every model in a 3D scene. When you move a model, it uses the coordinates



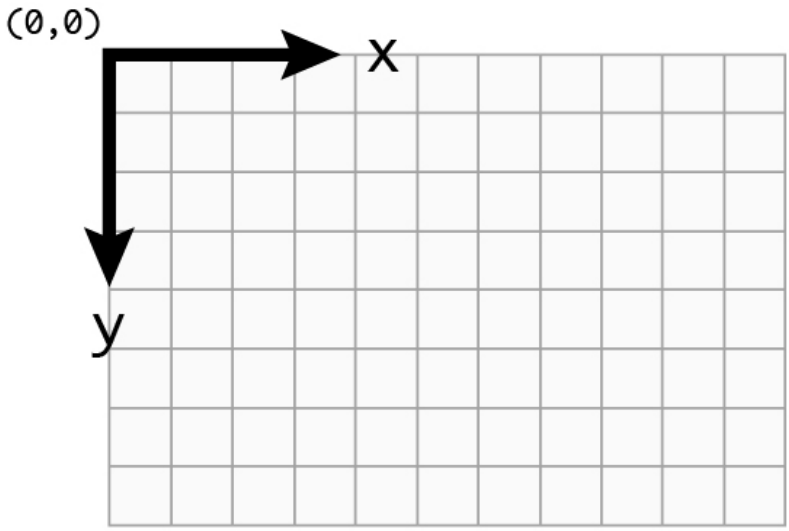
- Viewing Coordinate System:

  This coordinate system is based on the viewpoint of the observer, and changes as the "camera" moves.



- Screen Coordinate System:

  This coordinates system is the system used for the screen that displays something. It's a 2D system and is represented like this:



Sources: Coordinates Systems, Google Images

Question 4:

Three points A (1, 1); B (2, 3); C (3,2) to scale with ratio X = 2 and Y = 3.

Scale A:

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \times 1 + 0 \times 1 + 0 \times 1 + 0 \times 1 \\ 0 \times 1 + 3 \times 1 + 0 \times 1 + 0 \times 1 \\ 0 \times 1 + 0 \times 1 + 1 \times 1 + 0 \times 1 \\ 0 \times 1 + 0 \times 1 + 0 \times 1 + 1 \times 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 1 \\ 1 \end{bmatrix}$$

Scale B:

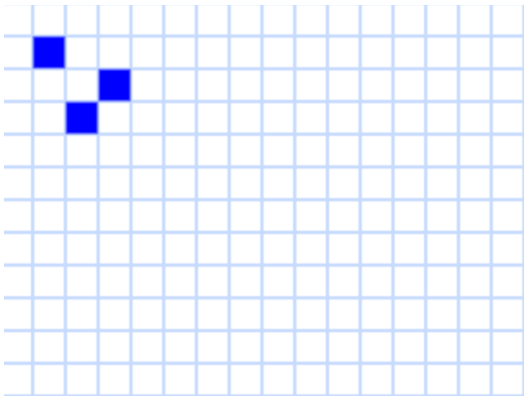
$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 2 \\ 3 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \times 2 + 0 \times 3 + 0 \times 1 + 0 \times 1 \\ 0 \times 2 + 3 \times 3 + 0 \times 1 + 0 \times 1 \\ 0 \times 2 + 0 \times 3 + 1 \times 1 + 0 \times 1 \\ 0 \times 2 + 0 \times 3 + 0 \times 1 + 1 \times 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 9 \\ 1 \\ 1 \end{bmatrix}$$

Scale C:

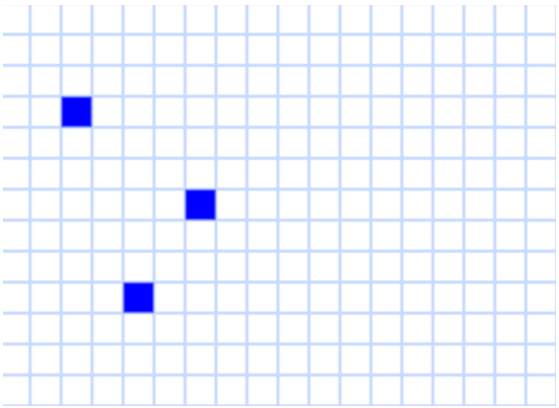
$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 3 \\ 2 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \times 3 + 0 \times 2 + 0 \times 1 + 0 \times 1 \\ 0 \times 3 + 3 \times 2 + 0 \times 1 + 0 \times 1 \\ 0 \times 3 + 0 \times 2 + 1 \times 1 + 0 \times 1 \\ 0 \times 3 + 0 \times 2 + 0 \times 1 + 1 \times 1 \end{bmatrix} = \begin{bmatrix} 6 \\ 6 \\ 1 \\ 1 \end{bmatrix}$$

Results: A (2,3); B (4, 9); C (6,6).

Before



After

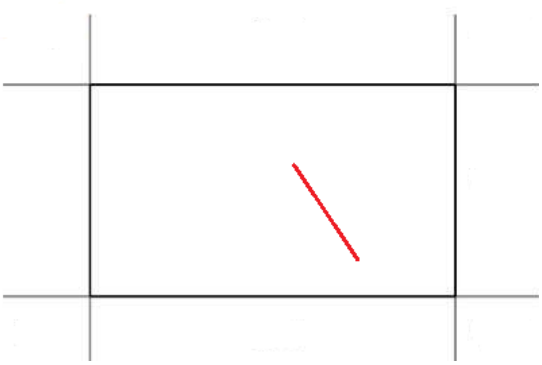


Sources: lectures, Matrix Calculator, Virtual Graph Paper

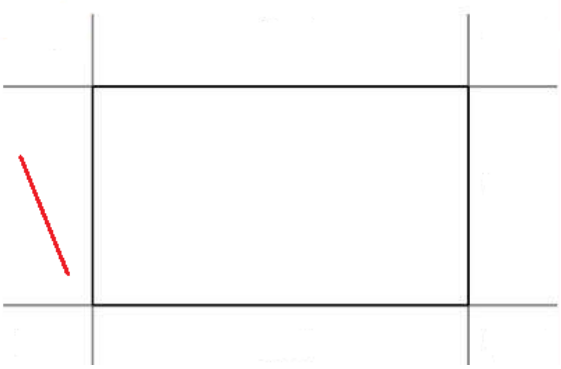
Question 5:

The Cohen-Sutherland Line Clippings Algorithm is used to remove unnecessary lines or portions of a line on a screen.

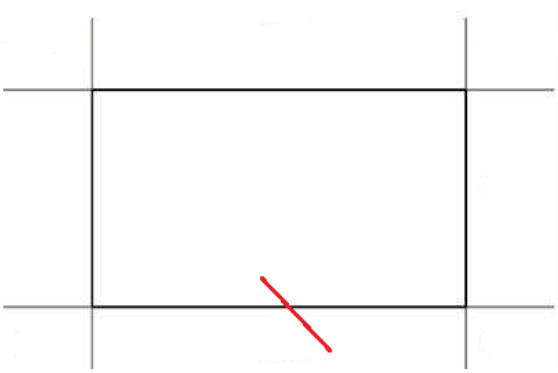
To explain the algorithm, I will go with 4 examples.



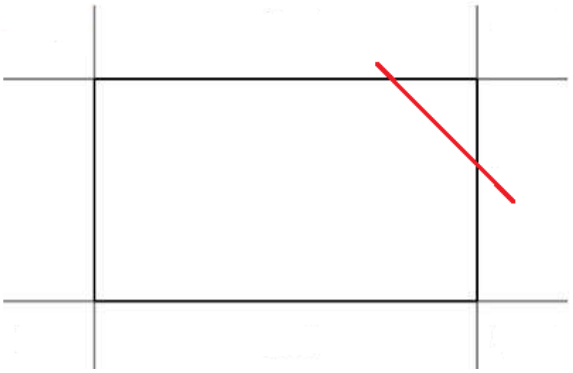
Line is inside



Line is outside

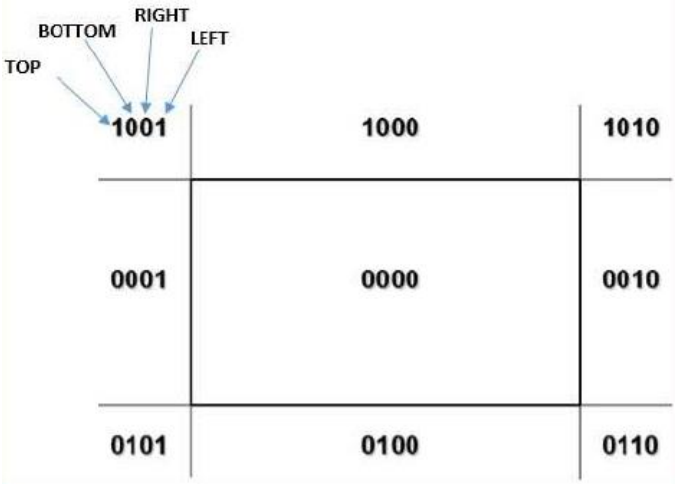


Line is partially in the screen



Line is crossing the screen

For each example, we use region code for each part we can distinguish:



First, the two easiest examples, when the line is inside and when the line is outside.

We identify in which region are located both endpoints of the line.

On the one where the line is inside, both endpoints are in the region 0000, on the one where the line is outside, both endpoints are in 0001.

Next step is to check if both endpoints are 0000. If it's true, we can accept the whole line.

So, in our example, we can accept the one where the line is inside.

Since none of the endpoints of the one outside is in 0000 we perform an AND operation on them. If the result is not 0000, we can reject the line.

Let's try: $0001 \text{ AND } 0001 = 0001$

The result is not 0000, so we reject that line.

So now, we will do it again with the line that is partially on the screen.

The endpoints are 0000 and 0100. They are not both on 0000 and the AND operation results in 0000, so we must clip the line.

We choose an endpoint that is outside of the screen, 0100 in our case. Then, we find the intersection point with the screen boundaries. This point will be the new endpoint of the line.

Next, we check for the other endpoint if it's 0000 or not. If it is, we accept the line with the new endpoint. If not, we also find the intersection point and create a new endpoint, then we accept the line.

In the case of the partial, we only have to cut the part that is in the 0100 region.

For the case of the crossing line, we have to cut the two endpoints.

Sources: Tutorials Point