Mode 7 Documentation

An inside view of the math behind it

Stanchi

July 31, 2019

Introduction

Mode 7 is a graphic mode developed by *Nintendo* in 1990, was used for the first time in the video game console Super NES, the 16 bits successor of the Nintendo Entertainment System. This one implemented, by hardware, the possibility of apply linear transformations to sprites, such as scaling, rotation, reflection, shearing and a composition of these with the positibility of adding a translation (being thus an *affine transformation*). It was one of the high points of the sale of this console in its time. This pseudo-3D graphic mode was mainly used for racing games.

The effect is very simple, it is based on projecting each pixel of a 2D texture to a 3D plane inclined a certain angle α relative to the origin of coordinates.

Mathematics

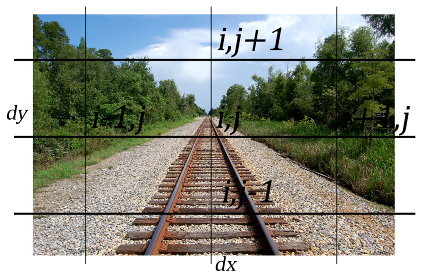
Mode 7 graphics are generated for each pixel by mapping background coordinates to screen coordinates using a linear transformation and setting the corresponding screen color.

Let $L : \mathbb{R}^3 \to \mathbb{R}^2$ be a linear transformation such that

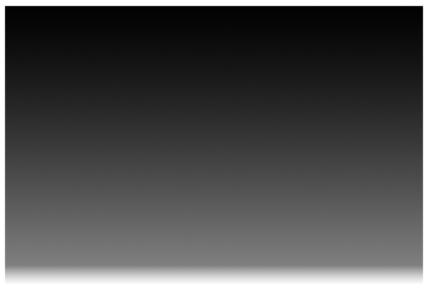
$$L\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \frac{x}{z} \\ \frac{y}{z} \end{pmatrix} = \begin{pmatrix} x^{'} \\ y^{'} \end{pmatrix}$$

and let be $z = \texttt{depth} + y$, what you will have is a linear transformation that trades height for depth, thus achieving a three-dimensional perspective. Note that the farthest points will be smaller, and viceverse.. Futhermore, we can asocciate to every pixel of our screen an index i and j.

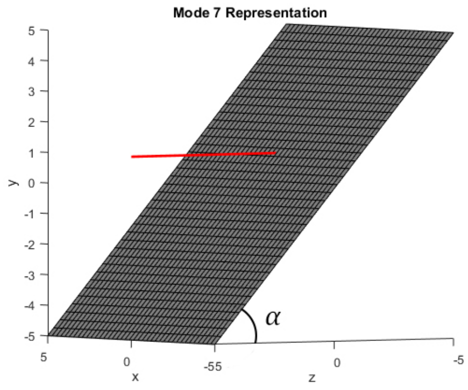
An example of all of this is shown in Figure 1.



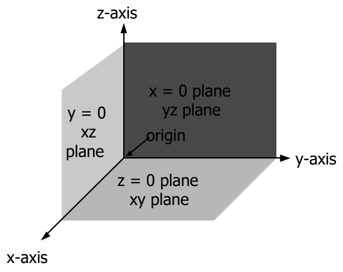
(a)



(b)



(c)



(d)

Figure 1: (a) Example of the farthest points that are smaller than the nearest ones. (b) Example of the gradient of distance from the subfigure (a) of railtracks, whitest part of the image are near points while the blackest part represents points far away. (c) Representation of how z is a dependent variable of y: the red line segment lenght will change depending on the y coordinate. (d) The right-handed Cartesian coordinate system indicating the coordinate planes.

Finally, we define `depth` as the max distance value between the screen (xy plane in first octant) and the gray plane (our texture), in addition, `depth` is a dependent variable of the angle α from the slope of the tangent vector to the gray plane.

Implementation

For the implementation we are using JAVA with Swing GUI library.

```java
BufferedImage bg = new BufferedImage(sWidth, sHeight,
    BufferedImage.TYPE_INT_RGB);
for (int j = 0; j < sHeight; j++) {
   for (int i = 0; i < sWidth; i++) {
      z = depth + j;

      y = ((sWidth / 2 - i) * Math.sin(Math.toRadians(angle)) + j *
          Math.cos(Math.toRadians(angle))) / z;
      y *= scale;
      y += y_offset;
      y %= Resources.background.getHeight();

      x = ((sWidth / 2 - i) * Math.cos(Math.toRadians(angle)) - j *
          Math.sin(Math.toRadians(angle))) / z;
      x *= scale;
      x += x_offset;
      x %= Resources.background.getWidth();

      bg.setRGB(i, j, Resources.background.getRGB((int) Math.abs(x),
          (int) Math.abs(y)));
   }
}
g.drawImage(bg, 0, 0, null);
```

For every pixel in our screen (i, j) we are:

- Setting z depending on the respective j.
- Appling different transformations.
- Setting each pixel of the texture (x, y) to the respective pixel in our screen (i, j).

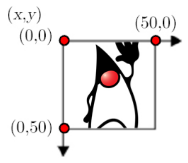


Figure 2: In JAVA, the screen coordinate x increases to the right and the screen coordinate y downwards.

We apply the composition of the following transformations:

$$R\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

$$L\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \frac{x}{z} \\ \frac{y}{z} \end{pmatrix}$$

$$S\begin{pmatrix} x \\ y \end{pmatrix} = \lambda\begin{pmatrix} x \\ y \end{pmatrix}$$

$$T\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}$$

$$M\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \bmod N \\ y \bmod N \end{pmatrix}$$

$$\Downarrow$$

$$(M \circ T \circ S \circ L \circ R)\begin{pmatrix} i \\ j \\ z \end{pmatrix} = M\left(T\left(S\left(L\left(R\begin{pmatrix} i \\ j \\ z \end{pmatrix}\right)\right)\right)\right)$$

We are focusing on the M transformation: we want our texture to repeat continuously in order to reach the '*infinity*'. Before we explain how to do so, it is important to consider some definitions from *modular arithmetic*.

Let be $f(x) = x^2 \pmod 2$, if we plot it:

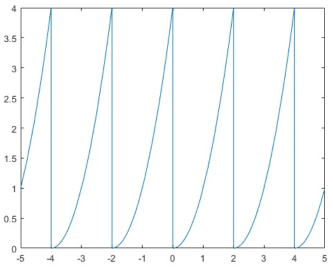


Figure 3: MATLAB Code: `fplot(@(x) mod(x,2).^2,[-5 5])`

As we can see, every 2 units of x the function repeats itself, that is because, in the vector space of mod 2, all possible values are in the interval $[0, 2)$, i.e:

$$x_1 = 3 \Longrightarrow \mathrm{mod}(x_1, 2) = \mathrm{mod}(3, 2) = 1$$

$$x_2 = 5 \Longrightarrow \mathrm{mod}(x_2, 2) = \mathrm{mod}(5, 2) = 1$$

$\therefore\ 3 \equiv 5 \pmod 2 \ \Rightarrow$ We have a congruence relation, so the value of $f(x)$ evaluated in 3 or 5 will be the same.

Taking this into consideration, we can define a function $p(x) = f(x \bmod N)$ which is a periodic function with period $N \Longrightarrow f(x) = f(x + N) \equiv p(x)$. Therefore the function $f(x)$ will repeat on intervals of length N. In the same way, the texture will repeat on intervals of its dimension.

References

[1] Grossman S, Flores J. Álgebra Lineal. 7^a ed. Ed McGraw-Hill. 978-607-15-0760-0 2012 (p.479)

[2] Churchill R, Brown J. Variable Compleja y Aplicaciones. 5^a ed. Ed McGraw-Hill. 0-07-010905-2

[3] Eckel B. Thinking in Java. 3^a ed. Prentice Hall. 0-13-027363-5

[4] Mode 7 (24/7/2019), https://en.wikipedia.org/wiki/Mode_7

[5] Tonc – GBA Programming (Mode 7) (24/7/2019), http://www.coranac.com/tonc/text/mode7.htm