A camera with a projection and view matrix

Drikus Kleefsman

January 25, 2010

Abstract

Keywords: Xna, world, projection, view, matrix

The first thing you want to have in a 3d scene is a camera to look at your world. Here is a program you can use to see if Xna is manipulating the matrices as you expected.

1 Introduction

Section 2 is the text of the Xna program. Section 3 gives the results as seen in the output screen.

2 The program

The program uses a very simple camera class, PerspectiveCamera. In it the View and Projection matrices are initialized using the CreateLookAt(...) and CreatePerspectiveFieldOfView(...) methods of the Xna struct Matrix.

The Matrix struct has 16 elements named M11..M44 and a large number of static methods. In the program a vector and a matrix are multiplied to show that in Xna the vector is a row and not a column. The a camera is placed at (0, 2, 0) that looks in the direction of (0, 0, 0) and has an up-vector in the -z direction. It is verified that the View matrix of the camera has the right elements by comparing these with the elements derived from the World matrix of the camera. And finally the projection matrix is applied to see what the values of vertices are in clip space and to see that the perspective division had not been done yet.

The theory of the rendering pipeline is the same for DirectX and OpenGL (of course). However, the formula's are different (OpenGL uses a column as a vector for example), so be careful translating software for OpenGL into that for DirectX!

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;
using D3Q.Cameras;

using SysDebug = System.Diagnostics.Debug;

namespace XnaMatrix1
{
    /// <summary>
    /// Version januari 2010; Drikus Kleefsman
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;
        PerspectiveCamera perspectiveCamera;

        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        /// <summary>
        /// Allows the game to perform any initialization it needs to before starting to run.
        /// This is where it can query for any required services and load any non-graphic
        /// related content.  Calling base.Initialize will enumerate through any components
        /// and initialize them as well.
        /// </summary>
        protected override void Initialize()
        {
            // TODO: Add your initialization logic here

            base.Initialize();
        }

        /// <summary>
        /// LoadContent will be called once per game and is the place to load
        /// all of your content.
        /// </summary>
        protected override void LoadContent()
        {
            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);

            //first, lets see how vector-matrix multiplication is done
            //Xna uses one row for a vector (instead of one column).
            //vector matrix multiplication is done as v.m and not
            //as m.v (because a vector is a row).
            //(OpenGL uses a column and m.v multiplication)

            //create a matrix (here element m11 gets value 1.1f etc):
```

```
            Matrix m = new Matrix(
                1.1f, 1.2f, 1.3f, 1.4f,
                2.1f, 2.2f, 2.3f, 2.4f,
                3.1f, 3.2f, 3.3f, 3.4f,
                4.1f, 4.2f, 4.3f, 4.4f)
                ;
            //and a vector4:
            Vector4 v = new Vector4(1.0f, 2.0f, 3.0f, 4.0f);
            //multiply en bring result to output window of debug (menu debug | windows | output):
            SysDebug.WriteLine("matrix: " + m);
            SysDebug.WriteLine("vector: " + v);
            SysDebug.WriteLine("product: "+Vector4.Transform(v, m));
            SysDebug.WriteLine("conclusion: indeed v.m and not m.v. \nFirst element product:");
            SysDebug.WriteLine("it is -- " + (v.X * m.M11 + v.Y * m.M21 + v.Z * m.M31 + v.W * m.M41));
            SysDebug.WriteLine("not   -- " + (m.M11 * v.X + m.M12 * v.Y + m.M13 * v.Z + m.M14 * v.W));
            SysDebug.WriteLine("\n");

            //the view matrix
            //it has the function of positioning a real camera

            //lets use a unusual aspect ratio 2.0 (x:y=2.0)
            const float aspect = 2.0f;
            perspectiveCamera = new PerspectiveCamera(this, aspect, new Vector3(0.0f, 2.0f, 0.0f),
                new Vector3(0.0f, 0.0f, 0.0f), new Vector3(0.0f, 0.0f, -1.0f));
            //Actually, the only interesting line in PerspectiveCamera is:
            //View = Matrix.CreateLookAt(position, position + forward, up);
            SysDebug.WriteLine("camera (view): " + perspectiveCamera.View);
            SysDebug.WriteLine("note: camera rotated -90 degrees around x-axis (M11 = 1, rest Mi1 = 0)");
            SysDebug.WriteLine("note: camera displaced in y direction (M41 2)");

            //-90?
            SysDebug.WriteLine("-90? Yes, because:");
            SysDebug.WriteLine("rotation around X: "+Matrix.CreateRotationX(MathHelper.PiOver2));
            SysDebug.WriteLine("note: M23=1 and M32=-1 (0,1,0,0)->(0,0,1,0) and (0,0,1,0)->(0,-1,0,0)");

            //World space coordiantes are multiplied bij the View matrix to get
            //the coordinates in Camera space.
            //Suppose the camera space coordinates are known as Vc. And we know
            //the World matrix W of the camera, then the coordinates in world space
            //Vw are: Vw = Vc.W
            //Multiply left and right by the inverse of W (_W) then Vw._W = Vc.
            //Meaning: the View matrix equals _W
            Matrix W = Matrix.CreateRotationX(-MathHelper.PiOver2)*
Matrix.CreateTranslation(new Vector3(0.0f, 2.0f, 0));
            SysDebug.WriteLine("world matrix: " + W);
            Matrix _W = Matrix.Invert(W);
            SysDebug.WriteLine("inverted W: "+ _W);
            SysDebug.WriteLine("indeed, View and _W are equal");
            SysDebug.WriteLine("equal Identity? " + W * _W);


            SysDebug.WriteLine("\n");
```

```
//now to the projection matrix
//it has the function of a lens in a real camera

//Actually, the only interesting line in PerspectiveCamera is:
//Projection = Matrix.CreatePerspectiveFieldOfView(
//    fov, aspectRatio, near, far);
SysDebug.WriteLine("camera (projection): " + perspectiveCamera.Projection);
SysDebug.WriteLine("You can see that M43 = -1, a translation over -near is implied");

//standard value of fov in PerspectiveCamera is:
//fov = MathHelper.PiOver4;
//fov (field of view) is in the Y-direction
//the near and far plane are:
//near = 1.0f; far = 1000.0f;
//just make a copy of these values:
const float fov = MathHelper.PiOver4;
const float near = 1.0f;
const float far = 1000.0f;

//calculate the coordinates of the near plane at distance 1.0
float ntop, nbottom, nleft, nright;
ntop = (float) Math.Tan(0.5f * fov) * near;
nbottom = - (float) Math.Tan(0.5 * fov) * near;
nleft = aspect * nbottom;
nright = aspect * ntop;
//and project all corners
Vector4 ntopLeft = new Vector4(nleft, ntop, -near, 1.0f);
Vector4 nbottomRight = new Vector4(nright, nbottom, -near, 1.0f);

SysDebug.WriteLine("near topleft: " + ntopLeft + "--> "+
    Vector4.Transform(ntopLeft, perspectiveCamera.Projection));
SysDebug.WriteLine("near bottomright: " + nbottomRight + "--> "+
    Vector4.Transform(nbottomRight, perspectiveCamera.Projection));

//calculate the coordinates of the far plane at distance 1000.0
float ftop, fbottom, fleft, fright;
ftop = (float)Math.Tan(0.5f * fov) * far;
fbottom = -(float)Math.Tan(0.5 * fov) * far;
fleft = aspect * fbottom;
fright = aspect * ftop;
//and project all corners
Vector4 ftopLeft = new Vector4(fleft, ftop, -far, 1.0f);
Vector4 fbottomRight = new Vector4(fright, fbottom, -far, 1.0f);

SysDebug.WriteLine("far topleft: " + ftopLeft + "--> " +
    Vector4.Transform(ftopLeft, perspectiveCamera.Projection));
SysDebug.WriteLine("far bottomright: " + fbottomRight + "--> " +
    Vector4.Transform(fbottomRight, perspectiveCamera.Projection));

SysDebug.WriteLine("note perspective divide: near plane W=1, far plane W=1000");
SysDebug.WriteLine("As you can see: vectors are projected to clip space, -1<X<1, -1<Y<1, 0<Z<1
// TODO: use this.Content to load your game content here
```

```
        }

        /// <summary>
        /// UnloadContent will be called once per game and is the place to unload
        /// all content.
        /// </summary>
        protected override void UnloadContent()
        {
            // TODO: Unload any non ContentManager content here
        }

        /// <summary>
        /// Allows the game to run logic such as updating the world,
        /// checking for collisions, gathering input, and playing audio.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing values.</param>
        protected override void Update(GameTime gameTime)
        {
            // Allows the game to exit
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
                this.Exit();

            // TODO: Add your update logic here

            base.Update(gameTime);
        }

        /// <summary>
        /// This is called when the game should draw itself.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing values.</param>
        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);

            // TODO: Add your drawing code here

            base.Draw(gameTime);
        }
    }
}
```

3 The output of the program

```
'XnaMatrix1.exe' (Managed): Loaded ....
...
matrix: { {M11:1,1 M12:1,2 M13:1,3 M14:1,4} {M21:2,1 M22:2,2 M23:2,3 M24:2,4}
  {M31:3,1 M32:3,2 M33:3,3 M34:3,4} {M41:4,1 M42:4,2 M43:4,3 M44:4,4} }
vector: {X:1 Y:2 Z:3 W:4}
product: {X:31 Y:32 Z:33 W:34}
conclusion: indeed v.m and not m.v.
First element product:
```

```
it is -- 31
not   -- 13


camera (view): { {M11:1 M12:0 M13:0 M14:0} {M21:0 M22:0 M23:1 M24:0}
 {M31:0 M32:-1 M33:0 M34:0} {M41:0 M42:0 M43:-2 M44:1} }
note: camera rotated -90 degrees around x-axis (M11 = 1, rest Mi1 = 0)
note: camera displaced in y direction (M41 2)
-90? Yes, because:
rotation around X: { {M11:1 M12:0 M13:0 M14:0} {M21:0 M22:-4,371139E-08 M23:1 M24:0}
     {M31:0 M32:-1 M33:-4,371139E-08 M34:0} {M41:0 M42:0 M43:0 M44:1} }
note: M23=1 and M32=-1 (0,1,0,0)->(0,0,1,0) and (0,0,1,0)->(0,-1,0,0)
world matrix: { {M11:1 M12:0 M13:0 M14:0} {M21:0 M22:-4,371139E-08 M23:-1 M24:0}
{M31:0 M32:1 M33:-4,371139E-08 M34:0} {M41:0 M42:2 M43:0 M44:1} }
inverted W: { {M11:1 M12:0 M13:0 M14:0} {M21:0 M22:-4,371139E-08 M23:1 M24:0}
      {M31:0 M32:-1 M33:-4,371139E-08 M34:0} {M41:0 M42:8,742278E-08 M43:-2 M44:1} }
indeed, View and _W are equal
equal Identity? { {M11:1 M12:0 M13:0 M14:0} {M21:0 M22:1 M23:0 M24:0}
  {M31:0 M32:0 M33:1 M34:0} {M41:0 M42:0 M43:0 M44:1} }


camera (projection): { {m11:1,207107 m12:0 m13:0 m14:0} {m21:0 m22:2,414213 m23:0 m24:0}
{M31:0 M32:0 M33:-1,001001 M34:-1} {M41:0 M42:0 M43:-1,001001 M44:0} }
You can see that M43 = -1, a translation over -near is implied
near topleft: {X:-0,8284271 Y:0,4142136 Z:-1 W:1}--> {X:-0,9999999 Y:0,9999999 Z:0 W:1}
near bottomright: {X:0,8284271 Y:-0,4142136 Z:-1 W:1}--> {X:0,9999999 Y:-0,9999999 Z:0 W:1}
far topleft: {X:-828,4271 Y:414,2136 Z:-1000 W:1}--> {X:-999,9999 Y:999,9999 Z:1000 W:1000}
far bottomright: {X:828,4271 Y:-414,2136 Z:-1000 W:1}--> {X:999,9999 Y:-999,9999 Z:1000 W:1000}
note perspective divide: near plane W=1, far plane W=1000
As you can see: vectors are projected to clip space, -1<X<1, -1<Y<1, 0<Z<1
The program '[5856] XnaMatrix1.exe: Managed' has exited with code 0 (0x0).
```

References

[1] Microsoft, “Xna Matrix class”, http://msdn.microsoft.com/en-us/library/microsoft.xna.framework.matrix_members.aspx