[Article: Simple 3D Camera in XNA](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

23[Comments](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx#comment)

**by Pete Street**

<http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx>

**How do I make a 3D camera?**

This is the most common question I hear from people starting out with 3D game development.  
There are many different types of possible cameras and various ways to achieve each of them. The amount of unique camera samples available online, while some are more complicated than others, can be daunting to those just beginning with 3D games.

Additionally, it’s frustrating and nearly impossible to combine two different samples without doing some serious hacking.  
My goal for this tutorial is to remove that frustration from 3D-beginners by providing a back-to-basics approach to three of the most common types of cameras used in games:

* a free camera
* a chase camera
* a orbit camera

The free camera will be able to yaw, pitch, and roll, and move in any direction.

The chase camera follows an object at a specified distance while also allowing for limited rolling and panning.

Finally, the orbit camera follows an object, can rotate any amount around it, and allows for limited rolling.

All three of these cameras are independent of one another and no other code relies on them. This way, it’s easy to add your own cameras later or modify one of mine without worrying about it changing something else.  
The focus for this tutorial will be on simplicity and getting your camera running with as little code as possible.  
Interspersed with the code examples, I’ll do my best to thoroughly explain what is going on – however, feel free to skip right to the example at any time.  
If you already have a decent understanding of cameras or need help on something specific, the code is structured such that, once the controls are altered to fit your needs, you should be able to plug it right into your existing game.

This tutorial assumes a basic understanding of C#, XNA, vector math, and world matrices. Nothing taught here is very complicated or math-heavy, but having an idea of how these work ahead of time will help.

**On to the Game!**

We’ll begin by starting a new XNA game project in Visual C#. I’ve named my project “SimpleCamerasTutorial,” but you may name yours whatever you wish.  
Once your project is created, add the two .fbx models that I’ve included, “ConeModel” and “CubeModel,” to the Content folder.

These models are for demonstration purposes only, as you’ll probably want to use much better-looking art assets in your actual game.

Now that we have everything in the project up and running, it’s time to begin coding.

**The Basic Camera**

The first thing to do is to add a Camera class. Right-click on your project and click Add ? Class, and name it “Camera.cs.” Then add the following Using statements to the top of the class:

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2  3 | using Microsoft.Xna.Framework;  using Microsoft.Xna.Framework.Graphics;  using Microsoft.Xna.Framework.Input; |

Now we’ll begin to add the class-level variables to Camera.cs. Type in the following fields:

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2  3 | private Vector3 position;  private Vector3 target;  public Matrix viewMatrix, projectionMatrix; |

These are the four variables that are absolutely necessary to a 3D camera. Position is the 3D coordinate of the camera, and target is the 3D coordinate of where the camera is looking at.

The viewMatrix is the “eye” in 3D space, and it takes in the position, target, and an Up vector to determine the camera’s position, where it’s looking, and which way is up.

The up vector will become important once we implement roll in the camera.

Finally, the projectionMatrix determines the camera’s properties like field-of-view, aspect ratio, and what distance in front of the camera to draw.

In essence, the projection matrix takes the 3D data from the view matrix and transforms (or projects) it onto the 2D screen.  
The View Matrix will need to be updated any time we change either the position, target, or up vector, but the Projection Matrix only needs to be defined once.

Next, add the constructor to Camera.cs:

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2  3  4 | public Camera()  {      ResetCamera();  } |

We’ll use the ResetCamera() method to define the initial values for the camera. Here’s ResetCamera():

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public void ResetCamera()  {      position = new Vector3(0, 0, 50);      target = new Vector3();        viewMatrix = Matrix.Identity;      projectionMatrix = Matrix.CreatePerspectiveFieldOfView(MathHelper.ToRadians(45.0f), 16 / 9, .5f, 500f);  } |

As you can see, the default values are set for the position, target, view matrix, and projection matrix.  
We’ve set the position to 50 units in front of the origin, initialized the view matrix to a standard identity matrix, and set the projection matrix to standard values.  
The first parameter in the projection matrix is field-of-view, which is usually at 45 degrees. Second is the aspect ratio, set to the 16:9 standard.  
The last two values in the projection matrix are the near plane and far plane, which are distances in front of the camera.  
The camera will only draw something if it is within these two boundaries. The target vector isn’t initialized to any values yet, because it will always depend on what type of camera you are using.

Next we’ll add an Update () method which will be called every frame and a private method called UpdateViewMatrix() to re-calculate the view matrix, which will be called from Update():

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public void Update()  {      UpdateViewMatrix();  }  private void UpdateViewMatrix()  {      viewMatrix = Matrix.CreateLookAt(position, target, Vector3.Up);  } |

UpdateViewMatrix() calculates a view matrix at the camera’s position, looking at the target, and with the standard up vector Vector3.Up using the Matrix.CreateLookAt() function.  
At the moment, the camera doesn’t move and its target and up vectors are fixed. We’re starting out with the most basic form of a view matrix to make sure that it draws properly.  
Later, this this code will be used as a base to add functionality for all of our cameras.

**The Basic Game**

That’s all we need for a basic, stationary camera. Now let’s move over to the Game1.cs class to make a sample to use the camera. Add the following variables to the top of Game1.cs:

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | Camera camera = new Camera();    Matrix cubeWorld;  Matrix coneWorld;    Model cubeModel;  Model coneModel; |

In Initialize(), set both of the world matrices to their identities:

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | protected override void Initialize()  {      cubeWorld = Matrix.Identity;      coneWorld = Matrix.Identity;        base.Initialize();  } |

Then in the LoadContent() method, load both of the models:

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2  3  4  5 | protected override void LoadContent()  {      cubeModel = Content.Load<Model>("CubeModel");      coneModel = Content.Load<Model>("ConeModel");  } |

In the Update() method, we’ll be updating the camera, and thus the camera’s view matrix:

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2  3  4 | protected override void Update(GameTime gameTime)  {      camera.Update();  } |

Finally, let’s draw both of the models using a private method named DrawModel(), and call that method in Draw():

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | protected override void Draw(GameTime gameTime)  {      //Draws both models      DrawModel(cubeModel, cubeWorld);      DrawModel(coneModel, coneWorld);  }    private void DrawModel(Model model, Matrix worldMatrix)  {      Matrix[] modelTransforms = new Matrix[model.Bones.Count];      model.CopyAbsoluteBoneTransformsTo(modelTransforms);        foreach (ModelMesh mesh in model.Meshes)      {          foreach (BasicEffect effect in mesh.Effects)          {              effect.EnableDefaultLighting();              effect.World = modelTransforms[mesh.ParentBone.Index] \* worldMatrix;              effect.View = camera.viewMatrix;              effect.Projection = camera.projectionMatrix;          }          mesh.Draw();      }  } |

The DrawModel() method takes in a model and its world matrix. If the model has multiple bones or many complex parts, this method then breaks them down to modelMeshes in order to be drawn with an effect. BasicEffect draws the scene here, but if you have a custom effect then feel free to use that instead. We have enabled the default lighting inside BasicEffect so there is shading on the model and not just an ambient light. The effect takes the world matrix of the model, and then the camera’s view and projection matrices. At this point, both the cone and the cube model should be set up and displayed correctly using the view and projection matrices from the camera. Press ‘F5’ to ensure that everything is working properly – you should see the front face of the cube with a cone sticking out of the top.

If everything looks correct, we can move on and make the cube a little more interesting with movement and rotation.

**Finishing the Game Sample**

We’ll now finish off the game sample to go along with the Camera class by adding rotation and movement to the cube. The cone will remain at the origin and serve as a point of reference, so we won’t need any fancy environments in order to know if the cube is moving. We already have the cube’s world matrix set up, and all we have to do is change it in Update().

Type the following code in the Update() method of Game1.cs:

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | protected override void Update(GameTime gameTime)  {      KeyboardState keyBoardState = Keyboard.GetState();        //Rotate Cube along its Up Vector      if (keyBoardState.IsKeyDown(Keys.X))      {          cubeWorld = Matrix.CreateFromAxisAngle(Vector3.Up, .02f) \* cubeWorld;      }      if (keyBoardState.IsKeyDown(Keys.Z))      {          cubeWorld = Matrix.CreateFromAxisAngle(Vector3.Up, -.02f) \* cubeWorld;      }        //Move Cube Forward, Back, Left, and Right      if (keyBoardState.IsKeyDown(Keys.Up))      {          cubeWorld \*= Matrix.CreateTranslation(cubeWorld.Forward);      }      if (keyBoardState.IsKeyDown(Keys.Down))      {          cubeWorld \*= Matrix.CreateTranslation(cubeWorld.Backward);      }      if (keyBoardState.IsKeyDown(Keys.Left))      {          cubeWorld \*= Matrix.CreateTranslation(-cubeWorld.Right);      }      if (keyBoardState.IsKeyDown(Keys.Right))      {          cubeWorld \*= Matrix.CreateTranslation(cubeWorld.Right);      }  } |

First, we use the ‘Z’ and ‘X’ keys to call Matrix.CreateFromAxisAngle(). This function rotates the cube along the standard up vector by .02 radians every frame. Then we move the cube using the arrow keys. Matrix.CreateTranslation() translates a world matrix by a vector. Instead of using standard vectors like Vector3.Forward, the vectors are taken directly from the cube’s world matrix. This way if we rotated the cube, all of its vectors rotated with it, and the cube moves whatever direction it’s facing rather than straight forward, back, left, and right. Press ‘F5’ again, and you should be able to rotate the cube and move it around.

**The Free Camera**

Now that we’ve finished with the basic game sample, we can work on the camera’s code, beginning with a 6-degrees-of-freedom camera. In Camera.cs, add the following variables and initialize them in ResetCamera():

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | private float yaw, pitch, roll;  private float speed;  private Matrix cameraRotation;    public void ResetCamera()  {      yaw = 0.0f;      pitch = 0.0f;      roll = 0.0f;        speed = .3f;        cameraRotation = Matrix.Identity;  } |

cameraRotation is the matrix we’ll be using to rotate the camera. Yaw, pitch, and roll are angles that will be passed in to cameraRotation, and speed is how fast the camera will move. Next, add a private method named HandleInput(), and call it in the Update() method:

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60 | public void Update()  {      HandleInput();      UpdateViewMatrix();  }    private void HandleInput()  {      KeyboardState keyboardState = Keyboard.GetState();        if (keyboardState.IsKeyDown(Keys.J))      {          yaw += .02f;      }      if (keyboardState.IsKeyDown(Keys.L))      {          yaw += -.02f;      }      if (keyboardState.IsKeyDown(Keys.I))      {          pitch += -.02f;      }      if (keyboardState.IsKeyDown(Keys.K))      {          pitch += .02f;      }      if (keyboardState.IsKeyDown(Keys.U))      {          roll += -.02f;      }      if (keyboardState.IsKeyDown(Keys.O))      {          roll += .02f;      }        if (keyboardState.IsKeyDown(Keys.W))      {          MoveCamera(cameraRotation.Forward);      }      if (keyboardState.IsKeyDown(Keys.S))      {          MoveCamera(-cameraRotation.Forward);      }      if (keyboardState.IsKeyDown(Keys.A))      {          MoveCamera(-cameraRotation.Right);      }      if (keyboardState.IsKeyDown(Keys.D))      {          MoveCamera(cameraRotation.Right);      }      if (keyboardState.IsKeyDown(Keys.E))      {          MoveCamera(cameraRotation.Up);      }      if (keyboardState.IsKeyDown(Keys.Q))      {          MoveCamera(-cameraRotation.Up);      }  } |

Finally, we put in another private method called MoveCamera():

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2  3  4 | private void MoveCamera(Vector3 addedVector)  {      position += speed \* addedVector;  } |

HandleInput() will take care of both the camera’s rotation and movement based on manual keyboard input (remember, this is just a sample, so you’re free to change the controls to whatever you like). Currently, the camera’s yaw is controlled by ‘J’ and ‘L,’ pitch is controlled by ‘I’ and ‘K,’ and roll is  
controlled by ‘U’ and ‘O.’ Similarly, the camera’s movement is mapped to ‘E’ and ‘Q’ moving up and down, ‘W’ and ‘S’ moving forward and back, and ‘D’ and ‘A’ moving left and right.

The MoveCamera() method alters the camera’s position in any direction at the camera’s speed. We use vectors from the rotation matrix instead of standard vectors like Vector3.Forward so if the camera is rotated at all, it will move in the direction that it’s facing. If you run the project right now, you should be able to move the camera around using the WASD + EQ keys, but you won’t be able to rotate the camera and it will always look at the same point. This is because cameraRotation isn’t affected by the yaw, pitch, and roll angles yet. To fix that, add the following code to the UpdateViewMatrix() method before you calculate the view matrix:

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | cameraRotation.Forward.Normalize();  cameraRotation.Up.Normalize();  cameraRotation.Right.Normalize();    cameraRotation \*= Matrix.CreateFromAxisAngle(cameraRotation.Right, pitch);  cameraRotation \*= Matrix.CreateFromAxisAngle(cameraRotation.Up, yaw);  cameraRotation \*= Matrix.CreateFromAxisAngle(cameraRotation.Forward, roll);    yaw = 0.0f;  pitch = 0.0f;  roll = 0.0f;    target = position + cameraRotation.Forward;    viewMatrix = Matrix.CreateLookAt(position, target, cameraRotation.Up); |

I make it a habit to normalize all the camera vectors that I’ll be using, and it’s really just a safeguard in case I accidentally edited it elsewhere. The first three lines of code normalize three of cameraRotation’s vectors, as those are the vectors that the camera will rotate around. The next three lines multiply cameraRotation by the Matrix.CreateFromAxisAngle() function, which rotates the matrix around any vector by a certain angle. We rotate the matrix around its own vectors so that it works properly no matter how it’s rotated already. After the matrix is rotated, the yaw, pitch, and roll values are set back to zero. Finally, the target is changed to accommodate the rotation matrix. It is set at the camera’s position, and then cameraRotation’s forward vector is added to it. This ensures that the camera is always looking in the direction of the forward vector, no matter how it’s rotated. Finally, we calculate the view matrix. The only change we made in this line is the last parameter – now the up vector is based on how the camera is rotated and not off the standard Vector3.Up. The view matrix needs an up vector to fully orient itself in 3D space, otherwise, the camera would have no way of knowing whether or not it’s upside-down.

Once this code is added, press ‘F5’ to run the program. Hooray! You should now have a free camera capable of moving in any direction, rotating in any direction, and independent of the cube!

**Expanding the Camera’s Horizons**

Now that we have a working free camera and the base of the Camera class, let’s make the class expandable so we can easily add new cameras in the future. Later, it will be simple to add the chase and orbit cameras, and any other cameras that you might come up with. To have multiple cameras within the same class, we’ll need an enum to keep track of which mode the camera is currently in. Add this code to the top of the Camera class:

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public enum CameraMode  {      free = 0,      chase = 1,      orbit = 2  }  public CameraMode currentCameraMode = CameraMode.free; |

The currentCameraMode will keep track of which camera mode is active. Right now, we only have the code for a free camera, so that is what we’ll set it to.

We also need to change the camera movement code in the HandleInput() method to make sure that it’s only run if the free camera is the current camera mode. The reason for this is the chase and orbit cameras’ positions don’t have their own movement code, and instead use the movement of an outside object. In HandleInput(), simply check if the free camera is currently active around the camera’s movement code:

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | private void HandleInput()  {      //Camera’s Rotation code        if (currentCameraMode == CameraMode.free)      {          //Camera’s Movement code.      }  } |

Then in UpdateViewMatrix(), place the code you just added for the free camera inside this switch block, keeping the view matrix calculation outside of it:

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | switch (currentCameraMode)  {      case CameraMode.free:            //Free-camera code goes here.            break;        case CameraMode.chase:            break;        case CameraMode.orbit:            break;  }    viewMatrix = Matrix.CreateLookAt(position, target, cameraRotation.Up); |

The goal is to keep all of the code simple, concise, and reusable, and as such the view matrix calculation should never change. Only the parameters it takes in should change.

**The Chase Camera**

Now it’s time to add the chase camera’s code. Add the following variables to the top of the Camera class and initialize them in the ResetCamera() method:

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | private Vector3 desiredPosition;  private Vector3 desiredTarget;  private Vector3 offsetDistance;    public void ResetCamera()  {      desiredPosition = position;      desiredTarget = target;        offsetDistance = new Vector3(0, 0, 50);  } |

The desired position and target will be used in conjunction with the actual position and target so we can get smooth transitions. A value that’s “desired” is what the actual values will constantly move to, but they don’t necessarily need to be equal to the desired value. The offset distance is how far displaced the camera is from the target. For both the chase and orbit cameras, the target won’t be defined by a vector in the rotation matrix, but rather by an outside object’s position – in this case, the target will be defined by the position of the cube. We’re going to add a matrix parameter to the Camera’s Update() method, and pass that matrix into the UpdateViewMatrix() method. In the game, we’ll pass the cube’s world matrix into camera.Update() in order to set that matrix as the one to chase. First change the Camera’s Update() and UpdateViewMatrix() methods like so:

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public void Update(Matrix chasedObjectsWorld)  {      UpdateViewMatrix(chasedObjectsWorld);  }    private void UpdateViewMatrix(Matrix chasedObjectsWorld)  {  } |

Then in Game1.cs, change camera.Update() to camera.Update(cubeWorld). Now we have a target to chase that isn’t dependent upon the camera’s code. Finally, for the actual chase camera’s code, add the following to the chase camera section of the switch block inside UpdateViewMatrix():

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | case CameraMode.chase:        cameraRotation.Forward.Normalize();      chasedObjectsWorld.Right.Normalize();      chasedObjectsWorld.Up.Normalize();        cameraRotation = Matrix.CreateFromAxisAngle(cameraRotation.Forward, roll);        desiredTarget = chasedObjectsWorld.Translation;      target = desiredTarget;      target.X += yaw;      target.Y += pitch;        desiredPosition = Vector3.Transform(offsetDistance, chasedObjectsWorld);      position = Vector3.SmoothStep(position, desiredPosition, .15f);        yaw = MathHelper.SmoothStep(yaw, 0f, .1f);      pitch = MathHelper.SmoothStep(pitch, 0f, .1f);      roll = MathHelper.SmoothStep(roll, 0f, .2f);        break; |

In this chase camera, I’ve allowed for a limited amount of yaw, pitch, and roll. This is to ensure that the camera always follows the target, but it will still have some degree of freedom on where to look. The first thing we do is normalize the rotation matrix’s forward vector because we’ll be using that vector to roll around. Notice that we don’t alter the rotation matrix with yaw and pitch like we did in the free camera. That’s because this camera’s yaw and pitch will depend on the target (not the rotation matrix), and that’s where desiredTarget comes in. Each frame, desiredTarget will be set to the position of whatever object we’re chasing (in this case the cube). We then set the actual target equal to the desiredTarget, and we can change the target’s X and Y coordinates at will. The yaw and pitch angles work well for this, as we can alter the target’s X to look a little left and right, and alter the target’s Y to look a little up and down.

Next we set the camera’s desiredPosition. The desired position could simply be set by adding the offset distance to the target, but what if the object we’re chasing rotates? Since we always want the camera positioned behind the object, desiredPosition needs to be transformed by the chased object’s world matrix.

Finally, we need to smooth the camera’s movement and transition the target vector back to the desired target. This is done with the SmoothStep() functions, which take in the current value, a desired value, and the factor that the current value is changed by. Using these methods, the position smoothly moves to desiredPosition, and the angles on which the camera rolls and pans on are constantly being moved back to zero.

At the top of Camera.cs, change currentCameraMode = CameraMode.free to currentCameraMode = CameraMode.chase. Now that the camera is in chase mode, you can run the project to ensure the chase camera works properly. You should be able to move and rotate the cube, and the camera should follow it smoothly. You can also pan the camera up, down, left, and right using the IJKL keys. In HandleInput(), you may want to increase the yaw and pitch values in order to make the camera panning more prominent, but you should be able to see the effect either way.

One fluke that you may notice after playing around with the chase camera is when you pan left or right and rotate the cube at the same time. You’ll see that the cube seems to be moving back and forth across your screen. The reason this happens has to do with how we define the target. Right now only the target’s X and Y coordinates are being edited, so it behaves much like a flat plane. When the camera rotates to the side of that flat plane, all of the points will appear to be in the center of the screen. As the camera rotates around it, the points in the plane will look like they’re inverted. Depending on the game, this may be a rather serious problem. Fortunately, it comes with a very simple solution, which is to multiply certain vectors in the chased object’s world matrix by the rotation angles so the target rotates with the cube. In UpdateViewMatrix(), replace these two lines:

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2 | target.X += yaw;  target.Y += pitch; |

…with these lines:

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2 | target += chasedObjectsWorld.Right \* yaw;  target += chasedObjectsWorld.Up \* pitch; |

Run the project again. You should be able to pan to the side and rotate the cube at the same time, and the target will rotate along with it.

**The Orbit Camera**

It’s now time to put the orbit camera in place. This will be especially easy to do since we already have all the necessary base code. We don’t need any more variables, but change the currentCameraMode at the top of the class to CameraMode.orbit. In UpdateViewMatrix(), inside the orbit section of the switch block, add the following code:

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | case CameraMode.orbit:    cameraRotation.Forward.Normalize();    cameraRotation = Matrix.CreateRotationX(pitch) \* Matrix.CreateRotationY(yaw) \* Matrix.CreateFromAxisAngle(cameraRotation.Forward, roll);    desiredPosition = Vector3.Transform(offsetDistance, cameraRotation);  desiredPosition += chasedObjectsWorld.Translation;  position = desiredPosition;    target = chasedObjectsWorld.Translation;    roll = MathHelper.SmoothStep(roll, 0f, .2f);    break; |

We start out by normalizing the rotation matrix’s forward vector, and then cameraRotation is calculated. The camera rolls using the same code that we’ve done in the past, rotating the matrix over its forward vector. What has changed is how the matrix takes in the yaw and pitch angles. Instead of yawing and pitching over cameraRotation’s vectors, we yaw and pitch over the world axes. By rotating over world axes instead of local axes, the orbiting effect is achieved. Next we transform the camera’s position by the offset distance and rotation matrix, much like what happened in the chase camera. At this point, the camera will orbit around the origin, but it won’t follow the cube if it moves. This is why the cube’s position is added to desiredPosition, so it will follow the cube and orbit around it no matter where it is. Since the target will always stay at the center of the cube, it is set to the cube’s position. Finally, the roll is smoothly brought back to zero.

Unlimited roll is not allowed in this sample for two reasons: 1) it’s disorienting and doesn’t look good, and 2) it can cause some stability issues. Because of floating-point precision, more roll than what we have right now will cause the camera to lose control and the game will become unplayable. However, as long as the amount of roll is contained, the camera is perfectly stable.

That’s all you need for the orbit camera! Run the game now and you should be able to move the cube and rotate around it. Something you may notice about the orbit camera’s code is that this line:

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1 | position = desiredPosition; |

…seems redundant. The reason I don’t simply transform the position is because you may want to add a SmoothStep() function to it like I did in the chase camera. I opted to leave it out of this sample, but you’re free to add it to your own code.

**Touching Up**

Congratulations! You made it through all the basic camera examples with (hopefully) little heartache. All there is left is to add a simple method to allow for easy switching between cameras. Add a public method named SwitchCameraMode() to Camera.cs:

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | public void SwitchCameraMode()  {      ResetCamera();        currentCameraMode++;        if ((int)currentCameraMode > 2)      {          currentCameraMode = 0;      }  } |

It’s a pretty simple method that loops through the camera modes. First, all of the camera’s default values are reset, then it cycles through the entire camera modes list. If you happen to add more camera modes in the future, just change the ‘2’ in this line:

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1 | if ((int)currentCameraMode > 2) |

…to the highest number in the enum at the top of the class. This method will be called in the game class, so head over to Game1.cs and type this line in the top of the class:

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1 | KeyboardState previousKeyBoardState = Keyboard.GetState(); |

Now simply add the following code to Update():

[?](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | protected override void Update(GameTime gameTime)  {      if (keyBoardState.IsKeyDown(Keys.Space) && previousKeyBoardState.IsKeyUp(Keys.Space))      {          camera.SwitchCameraMode();      }        //This line should be at the bottom of Update()      //after all of the input code.      previousKeyBoardState = keyBoardState;    } |

That’s it! You now have a simple, working sample with three distinct camera modes which you can change on-the-fly.

**The Next Step**

This tutorial contains simple examples of what you can do with 3D cameras. Everything is reliable and designed to be easy to use, but it’s also somewhat primitive. Using these examples as a launch pad, you can improve the cameras by adding more sophisticated functions to them. For instance, most chase camera samples (including the one in XNA’s education catalog) use a physics-based spring system for smoothly updating the position. We faked a spring by using the SmoothStep() functions, but if you need something more advanced, you might consider making one yourself.

You may also want to make different types of cameras. These three examples are among the most common in games and can be adapted to fill many needs, but if your needs aren’t met, then you have the base code to easily make a different camera.

If you want to save fiddling with your camera for later and want to get started with the game right now, this sample is easy to integrate with your game. All you have to do is add the camera class to your own game project, and initiate and update the camera in the game as I’ve shown above. Right out of the box, you’ll have three different cameras to use for developing your game.

**Closing Words**

Throughout this tutorial, I’ve made an effort to keep everything simple and to explain all of the code in detail. Again, my goal is to give people starting out with 3D an easy-to-understand introduction into 3D cameras. This will hopefully curb some of the frustration that comes out of learning something new, and will allow you to focus on the rest of your game more quickly and easily. Thanks for reading, and keep making games that are absolutely fine!

[Download Sample Project](http://www.madgamedev.com/Downloads/articles/SimpleCamerasTutorial.zip)

By [10](http://www.madgamedev.com/author/10.aspx) on September 5, 2010 at 9:09 AM   
Tagged:

**Related posts**

[Article: Vacant Skies &#8211; Action RPG Tutorial Series](http://www.madgamedev.com/post/2010/09/18/Article-Vacant-Skies-8211;-Action-RPG-Tutorial-Series.aspx)by Aaron T Foley Welcome to the Vacant Skies – Action RPG Tutorial Series. My name is Aaron Fole...[Article: Programming your first XNA 4.0 game for PC, Xbox 360 &amp; Windows Phone 7: Pong](http://www.madgamedev.com/post/2010/08/24/Article-Programming-your-first-XNA-40-game-for-PC-Xbox-360-amp;-Windows-Phone-7-Pong.aspx)by Thomas "Mister Helmut" Altenburger I’d like to introduce the XNA framework to you with this ve...[Article: Rolling World Tutorial](http://www.madgamedev.com/post/2010/09/08/Article-Rolling-World-Tutorial.aspx)by Christian Schlager When you look around the XBox Indie Games Channel, there aren't a lot o...

**Comments (23)** [**-**](javascript:toggle_visibility('commentlist',%20'commenttoggle');)

[Reply](javascript:void(0);)



**Alejandro**   
9/6/2010 5:09:12 AM [#](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx#id_eb861b52-fab4-4fa2-91f8-414d9b0f75d5)

Very nice/concise article!.   
I would like to point out a neat trick that I read somewhere, sadly I can't remember from whom...   
  
It's related to the enums.   
As long as the enums are sequentially enumerated (starting from zero, and incrementing one by one, the default behaviour) you can declare one last enum, ideally named "Count":   
  
enum SomeEnum   
{   
      a,   
      b,   
      c,   
      d,   
      Count   
}   
  
these will have these implicit int values:   
  
enum SomeEnum   
{   
     a = 0,   
     b = 1,   
     c = 2,   
     d = 3,   
     Count = 4   
}   
  
The SwitchCameraMode() branch could be changed to this:   
  
if ((int)currentCameraMode > (int)CameraModes.Count - 1)   
{   
       currentCameraMode = 0;   
}   
  
(or any other variant, like if(currentMode == CameraMode.Count), etc).   
  
As long as new camera modes are added before "Count", this code would never need to be watched over.   
Very useful also on complex state machines.

[Reply](javascript:void(0);)



**Pete Street**   
9/6/2010 1:09:12 PM [#](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx#id_19fe3ec4-8434-4192-8062-e27327c81425)

Thanks for the compliment, Alejandro!   
  
That's a very clever trick with enums that I hadn't thought of before, and would make working with them a little less tedious.   
  
If anyone else has any comments/criticisms/questions/suggestions, I'm more than happy to answer them!

[Reply](javascript:void(0);)



**Pieter**   
9/20/2010 6:09:09 PM [#](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx#id_34885959-3ffd-4bcf-9721-44bdb8cd8d59)

Amazing tutorial!! Very easy to follow and very informative!

[Reply](javascript:void(0);)



[**Pete Street**](http://petestreetproductions.blogspot.com/)   
9/24/2010 12:09:22 AM [#](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx#id_6ff2cedd-a2ca-4782-a9d8-9c955c251cab)

Thanks, Pieter!

[Reply](javascript:void(0);)



**Ubaga**   
11/11/2010 9:11:36 AM [#](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx#id_329b9fa7-8691-47b1-a34e-5a563f64667e)

I have a problem with rendering.Whenever i move/rotate camera it renders the model again and again, so i end up with several default objects :/ Could you point me in right direction how to fix it? Thanks

[Reply](javascript:void(0);)



[**Pete Street**](http://petestreetproductions.blogspot.com/)   
11/16/2010 1:11:38 AM [#](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx#id_77e83276-fefd-4870-a7c4-c16f80bda8be)

@Ubaga   
  
What does your relevant code look like?  Are you using this sample or integrating the camera into your own game?   
  
The rendering code is very standard and I pulled it off of MSDN to keep things simple, so the problem might be somewhere else.     
  
Hopefully I can help...

[Reply](javascript:void(0);)



**Jazz**   
12/14/2010 8:12:13 PM [#](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx#id_ba27480f-cb44-47bc-914a-c07678491606)

Shouldn't the elapsed time between two calls to Update() be used when modifying rotations or position? Else camera movement speed wont be consistent  and will vary depending how fast the application is running.

[Reply](javascript:void(0);)



[**Pete Street**](http://petestreetproductions.blogspot.com/)   
12/24/2010 4:12:31 AM [#](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx#id_6c483212-5ec0-4617-a830-5917c297b2d0)

@Jazz If you aren't using a fixed timestep in the game, that may be true.  In my XNA games, however, I generally stick to the default 60 fps because it keeps things easy.

[Reply](javascript:void(0);)



**Jazz**   
12/30/2010 11:12:35 PM [#](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx#id_d5bdb195-40e6-47a4-8a30-39c78a9790c6)

yeh, I dont like to use a fix timestep or synchronized vertical retrace when debugging so I can see the true fps. If anyone interested, I just add in the Handle Input for FreeCam mode:   
  
float elaspedTime = (float)gameTime.ElapsedGameTime.TotalSeconds;   
  
//Do this for rotations   
yaw += rotateSpeed \* elaspedTime;   
  
//Do this for moving   
Move(rotation.Forward \* elaspedTime);   
  
//Change Move Function   
position += moveSpeed \* direction;   
  
Great Tutorial!! Very Easy to build on. Thanks!

[Reply](javascript:void(0);)



**Dafvid**   
1/13/2011 2:01:07 AM [#](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx#id_016d085a-c1d2-4584-b27f-b30b4bf3745b)

Thank you so much for this tutorial! It really helped me create the camera the way I wanted it. Got a suggestion not to use tirgonometry in 3D but to use vectors and matrices instead, this is the way to do it for sure.   
  
I've made my own modification of the rotation. Instead of keeping track of yaw, roll and pitch. I've removed roll, since it's confusing in any other context than flying. And I just rotate the matrix directly. Also I'm checking to see if we're looking straight up or straight down, in that case, stop rotating. Going a full circle in forward rotation also feels "unnatural" for humans in a normal context.   
  
  
public enum Direction { Up, Down, Left, Right, Forward, Backward }   
  
private const float \_ROTATION = (float)(Math.PI / 75);   
  
public void Rotate(Direction direction)   
        {   
            //Debug.WriteLine("Rotating " + direction);   
            Debug.WriteLine(Vector3.Dot(Matrix.Identity.Up, \_cameraMtx.Forward));   
            if (direction == Direction.Up)   
                if(Vector3.Dot(Matrix.Identity.Up,\_cameraMtx.Forward)  -0.99f) //stops camera facing down   
                    \_cameraMtx \*= Matrix.CreateFromAxisAngle(-\_cameraMtx.Right, (float)(\_ROTATION));   
  
            if (direction == Direction.Left)   
                \_cameraMtx \*= Matrix.CreateRotationY((float)(\_ROTATION));   
            if (direction == Direction.Right)   
                \_cameraMtx \*= Matrix.CreateRotationY((float)(-\_ROTATION));   
        }

[Reply](javascript:void(0);)



**youri**   
3/17/2011 2:03:27 PM [#](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx#id_506f962a-af95-4274-9614-c1399fd0b3b8)

lovely tutorial mate.   
had lots of code from my teacher but didn't understand anything from it, but this was very well explained along the way.

[Reply](javascript:void(0);)



**Hatchy**   
3/17/2011 3:03:37 PM [#](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx#id_322b2298-d021-4c80-8082-0dba99d2b055)

At the part where it says;   
  
protected override void Update(GameTime gameTime)   
{   
  KeyboardState keyBoardState = Keyboard.GetState();   
  
  //Rotate Cube along its Up Vector   
  if (keyBoardState.IsKeyDown(Keys.X))   
  {   
    cubeWorld = Matrix.CreateFromAxisAngle(Vector3.Up, .02f) \* cubeWorld;   
  }   
  if (keyBoardState.IsKeyDown(Keys.Z))   
  {   
    cubeWorld = Matrix.CreateFromAxisAngle(Vector3.Up, -.02f) \* cubeWorld;   
  }   
  
  //Move Cube Forward, Back, Left, and Right   
  if (keyBoardState.IsKeyDown(Keys.Up))   
  {   
    cubeWorld \*= Matrix.CreateTranslation(cubeWorld.Forward);   
  }   
  if (keyBoardState.IsKeyDown(Keys.Down))   
  {   
    cubeWorld \*= Matrix.CreateTranslation(cubeWorld.Backward);   
  }   
  if (keyBoardState.IsKeyDown(Keys.Left))   
  {   
    cubeWorld \*= Matrix.CreateTranslation(-cubeWorld.Right);   
  }   
  if (keyBoardState.IsKeyDown(Keys.Right))   
  {   
    cubeWorld \*= Matrix.CreateTranslation(cubeWorld.Right);   
  }   
}   
  
I get an error on each line saying     
"Microsoft.XNA.Framework.Graphics.Model does not contatin a definition for 'Forward' and no extension method 'Forward' accepting a first argument of type 'Microsoft.Xna.Framework.Graphics.Model' could be found."   
I've copied your tutorial line for line up to this point. When I go to debug the program, my model does not move at all. What have I done wrong?

[Reply](javascript:void(0);)



**PeteStreet**   
6/5/2011 1:06:00 PM [#](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx#id_26d07287-d566-4984-ac8e-e3576a616430)

Hey Hatchy,   
  
Sorry that I haven't checked this in a while (and the comments haven't worked) Tong  
  
If you still need help (or someone else reading this) here you go.  I'm guessing that you're referring to this line:   
  
cubeWorld \*= Matrix.CreateTranslation(cubeWorld.Forward);   
  
Based on the error, I assume that in your code you're writing "cubeModel" instead of "cubeWorld."  "cubeWorld" is the world matrix for the cube, which includes a "Forward" vector.     
  
Just make sure that you're applying a Matrix operation (Matrix.CreateTranslation()) to a matrix (cubeWorld) and not a model (cubeModel).  You can think of the model as the visual representation of its world matrix, not the matrix itself.   
  
If that didn't help, let me know!

[Reply](javascript:void(0);)



**Nickolai51**   
8/8/2011 11:08:38 AM [#](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx#id_c02065bd-d59e-4945-8aec-034c4a540f99)

Hi, great tutorial.. very helpful Smile  
it all works fine with the models you provided, however, when I try to use my models, which I know normally work, they dont appear, could you suggest why this is and how I could fix it?

[Reply](javascript:void(0);)



**Nickolai51**   
8/8/2011 1:08:58 PM [#](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx#id_ebc43de3-679d-42e7-bd0a-04ad856d7950)

haha Smileforget it, it was my model exporting at fault \*facepalm\*

[Reply](javascript:void(0);)



**Mobildick**   
5/3/2012 12:05:16 AM [#](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx#id_bb0e5445-882d-474f-9392-2710cb04c38a)

Very nice tutorial thanks man. But ive got one problem ive been working on xna and c# since 2weeks and anytime i finish writing the camera my car model always appear big and also rest ontop of the camera making me see the some bottom of the car. Ive tried all i could but no sucess can u please help.

[Reply](javascript:void(0);)



**Rafael**   
5/8/2012 10:05:09 PM [#](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx#id_b2b7bb6f-7ef8-4bfe-8383-f24c788c4e7f)

Can't download the attachment, you can fix it?   
  
Thanks

[Reply](javascript:void(0);)



**Mobildick**   
5/10/2012 8:05:27 PM [#](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx#id_50e88577-da29-42d7-922e-c86ad42a1044)

please how can i moved my carmodel while on a chase camera ive tried all i could but no success, please help me.

[Reply](javascript:void(0);)



**Mobildick**   
5/10/2012 8:05:44 PM [#](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx#id_1290e275-a3a8-4f7d-ad99-658b055c89e7)

when i press the up n down key the camera moves but the car stay still please help this has been killing me for days. am using the code above for the camera n its place right behind the car as i want but car stay still while camera moves.

[Reply](javascript:void(0);)



**turbon**   
6/24/2012 7:06:34 PM [#](http://www.madgamedev.com/post/2010/09/05/Article-Simple-3D-Camera-in-XNA.aspx#id_405107b2-391b-4225-979c-da5cb2748e8f)

Attachement - cant download. And what is "chasedObjectsWorld" ?