

## Adding a Skybox

### Simple Object Creation with our Existing Classes

The first easy object that we can put together with what we've got going for us right now is a skybox. This will be a reusable object for almost any project we come up with later on, whether it is an outdoor shooter or an indoor puzzle game that just happens to have windows every now and then. The concept behind a skybox is well understood, but for some simple review I'll go over it here in case anyone is brand new to this wide world of game development.

### Skyboxes in a Nutshell (or, reasons for this week's code changes)

The basic idea behind a skybox is incredibly simple. We simply render a cube (sometimes a half sphere, which is then called a sky dome) inside out with some pretty scenic or other fitting textures on its surfaces. Now, this alone will not make it a skybox. Skyboxes are also usually incredibly large (farther away in the world than anything we want to be seen at least, which is sometimes done more with rendering math than actual size) and also tend to follow the movement of the camera. By forcing the center of the cube that makes up the skybox to be at the camera's current position at each frame, we are able to make the scene rendered on the inside of the box look like it is infinitely far away since far objects seem to move slower than near ones, and the skybox won't be moving at all relative to our eyes.

### Putting the Skybox Class together

After all the not so simple shader and camera handling code we have written, the skybox will be a fairly simple endeavor. There are of course a couple of ways to put together a skybox. The method I used in the MDX 1.1 version of the engine was simply a group of textured quads (which at that time were passed their vertices and indices and didn't necessarily have to be quads) that were rendered in a simple for loop. The method used here will be similar, except we are no longer going to be passing the vertices to the class at creation (since we are assuming it will always be a group of quads). Instead, we'll set the rotations for the sides of the box and keep track of the offsets each side has from the center:

```
using HMEngine.HMCameras;
using HMEngine.HMEffects;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;

namespace HMEngine.HMObjects {
    public class HMSkybox : HMObject {
        private readonly Color myColor;
        private readonly string[] myFiles;
        private readonly Vector3[] myOffsets;
        private readonly HMQuad[] mySides;

        public HMSkybox(string[] textures, Color color) {
            myFiles = textures;
            mySides = new HMQuad[6];
            myOffsets = new Vector3[6];
        }
    }
}
```

### Tutorial 5 - Adding a Skybox

```

myColor = color;
Scaling = new Vector3(5);

CreateSides();
}

```

Our implementation consists of an array of sides, the texture files for each side, the offsets of positions of each side away from center, and a color to use for rendering with the textures. The textures are loaded inside the textured quads, so all we will have to do is pass the files into their constructors inside CreateSides. The offsets will be calculated based on the scaling of the whole skybox, which for now I am setting up as 5 in all directions just to get past our near clipping plane. Here is the load content function and the ones for setting up the sides and offsets:

```

private void CreateSides() {
    for (int i = 0; i < 6; i++) {
        mySides[i] = new HMQuad(myFiles[i], myColor) { Scaling = Scaling };
    }

    mySides[0].Rotation = Quaternion.CreateFromAxisAngle(new Vector3(1, 0, 0), MathHelper.PiOver2);
    mySides[1].Rotation = Quaternion.CreateFromAxisAngle(new Vector3(1, 0, 0), -MathHelper.PiOver2);
    mySides[2].Rotation = Quaternion.CreateFromAxisAngle(new Vector3(0, 1, 0), MathHelper.PiOver2);
    mySides[3].Rotation = Quaternion.CreateFromAxisAngle(new Vector3(0, 1, 0), -MathHelper.PiOver2);
    mySides[5].Rotation = Quaternion.CreateFromAxisAngle(new Vector3(0, 1, 0), MathHelper.Pi);

    CalculateOffsets();
}

private void CalculateOffsets() {
    myOffsets[0] = new Vector3(0, 0.5f, 0) * Scaling;
    myOffsets[1] = new Vector3(0, -0.5f, 0) * Scaling;
    myOffsets[2] = new Vector3(-0.5f, 0, 0) * Scaling;
    myOffsets[3] = new Vector3(0.5f, 0, 0) * Scaling;
    myOffsets[4] = new Vector3(0, 0, -0.5f) * Scaling;
    myOffsets[5] = new Vector3(0, 0, 0.5f) * Scaling;
}

public override void LoadContent(GraphicsDevice myDevice, ContentManager myLoader) {
    foreach (HMQuad quad in mySides) {
        quad.LoadContent(myDevice, myLoader);
    }

    VertexDeclaration = mySides[0].VertexDeclaration;
}

```

Now, the only thing we have left to get our skybox working is the code for rendering it. Since the setting up of object specific parameters needs to be done for each side of the box, we will call that function in our rendering loop. This will handle setting the parameters for all the child quad objects for us:

```

public override void Render(GraphicsDevice myDevice) {
    myDevice.RenderState.DepthBufferWriteEnable = false;
    for (int i = 0; i < 6; i++) {
        mySides[i].Position = HMCameraManager.ActiveCamera.Position + myOffsets[i];
        HMEffectManager.ActiveShader.SetParameters(mySides[i]);
        mySides[i].Render(myDevice);
    }
    myDevice.RenderState.DepthBufferWriteEnable = true;
}

public override void UnloadContent() { }
}

```

The Render function simply loops through all six sides of the box, updates their current positions to be centered around the current camera position, and passes the updated information to the shader before rendering the actual object. We also turn off depth buffer writing so it is always behind other objects.

We will need to make sure to always add the skybox to the object manager before anything else for this to work. That is all we should need to get a skybox up and going. To the HMDemo, I added a folder to our content that contains the 6 textures for the sides of the skybox and named them top, bottom, left, right, front, and back. Once this is done all that is needed is to create and add the object with the list of textures and to run the demo. Here is the new version of the demo code with the added skybox:

```
using System.Collections.ObjectModel;
using HMEngine;
using HMEngine.HMCameras;
using HMEngine.HMEffects;
using HMEngine.HMInputs;
using HMEngine.HMObjects;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;

namespace HMDemo {
    internal static class HMDemo {
        private static readonly HMGame game = new HMGame(800, 600);
        private static readonly HMQuad quad = new HMQuad("Content/Textures/hazymind", Color.White);
        private static readonly HMShader shader = new HMShader("HMContent/Shaders/TransformColorTexture");
        private static readonly HMKeyboardDevice keyboard = new HMKeyboardDevice();
        private static readonly HMMouseDevice mouse = new HMMouseDevice();
        private static readonly HMGamePadDevice gamepad = new HMGamePadDevice(PlayerIndex.One);

        private static readonly string[] sky = {
            "Content/Textures/Skybox/top",
            "Content/Textures/Skybox/bottom",
            "Content/Textures/Skybox/left",
            "Content/Textures/Skybox/right",
            "Content/Textures/Skybox/front",
            "Content/Textures/Skybox/back"
        };

        private static readonly HMSkybox skybox = new HMSkybox(sky, Color.White);

        public static void Main() {
            HMInputManager.AddDevice("keyboard", keyboard);
            keyboard.OnKeyReleased += keyboard_OnKeyReleased;

            HMInputManager.AddDevice("mouse", mouse);
            mouse.OnMouseMoved += mouse_OnMouseMoved;
            mouse.OnMouseScrolled += mouse_OnMouseScrolled;

            HMInputManager.AddDevice("player1", gamepad);
            gamepad.OnButtonReleased += gamepad_OnButtonReleased;
            gamepad.OnJoystickMoved += gamepad_OnJoystickMoved;

            HMEffectManager.AddEffect("TCT", shader);
            skybox.Shader = "TCT";
            quad.Shader = "TCT";

            HMObjectManager.AddObject("skybox", skybox);
            HMObjectManager.AddObject("quad", quad);
            game.Run();
        }

        private static void keyboard_OnKeyReleased(Collection<Keys> keys) {
            if (keys.Contains(Keys.F)) {
                game.ToggleFullScreen();
            }
        }
    }
}
```

```
        if (keys.Contains(Keys.Escape)) {
            game.Exit();
        }
    }

    private static void mouse_OnMouseMoved(Vector2 move) {
        HMCameraManager.ActiveCamera.Revolve(new Vector3(1, 0, 0), move.Y * 0.01f);
        HMCameraManager.ActiveCamera.RevolveGlobal(new Vector3(0, 1, 0), move.X * 0.01f);
    }

    private static void mouse_OnMouseScrolled(int ticks) {
        HMCameraManager.ActiveCamera.Translate(new Vector3(0, 0, ticks * 0.01f));
    }

    private static void gamepad_OnButtonReleased(Collection<HMGamePadButton> buttons) {
        if (buttons.Contains(HMGamePadButton.Back)) {
            game.Exit();
        }
    }

    private static void gamepad_OnJoystickMoved(Vector2 leftStick, Vector2 rightStick) {
        HMCameraManager.ActiveCamera.Revolve(new Vector3(1, 0, 0), leftStick.Y * 0.1f);
        HMCameraManager.ActiveCamera.RevolveGlobal(new Vector3(0, 1, 0), leftStick.X * 0.1f);
        HMCameraManager.ActiveCamera.Translate(new Vector3(0, 0, -rightStick.Y * 0.5f));
    }
}
}
```