Lab 7 – All Together Now

Introduction

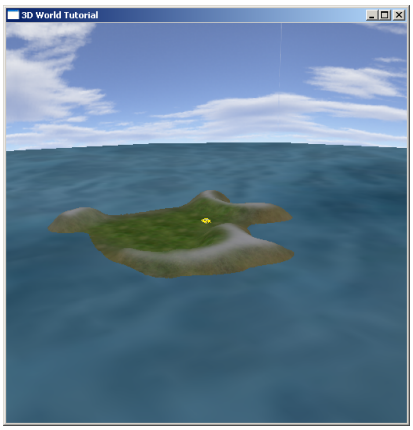
In the previous six OpenGL tutorials we have learned to use programmable shaders, textures, shadow maps, render to texture, shading and cube mapping. You also learned to implement a simple light model, some camera control and animation. The techniques have been demonstrated with very simple programs.

It is time to put this together into something more useful. In this final OpenGL tutorial we will get to try a more realistic scenario, we will:

- Try out the 3D Studio MAX modeling package.
- Export OBJ files.
- Enhance a very simple 3D application using the techniques we have learned.

The idea is to give you a feel for how workflow with 3D modeling and real time rendering works. Most of the data used in games and other 3D applications has been hand crafted in modeling packages such as MAX or Maya. OBJ is a very basic file format, but is pretty much universally supported and is a good place to start.

The example program we will work with can –at this point- load OBJ models, it has a sun that orbits the (flat) earth, consisting of a little island. Now is a good time to check it out: run the example program. Not very stylish, is it?



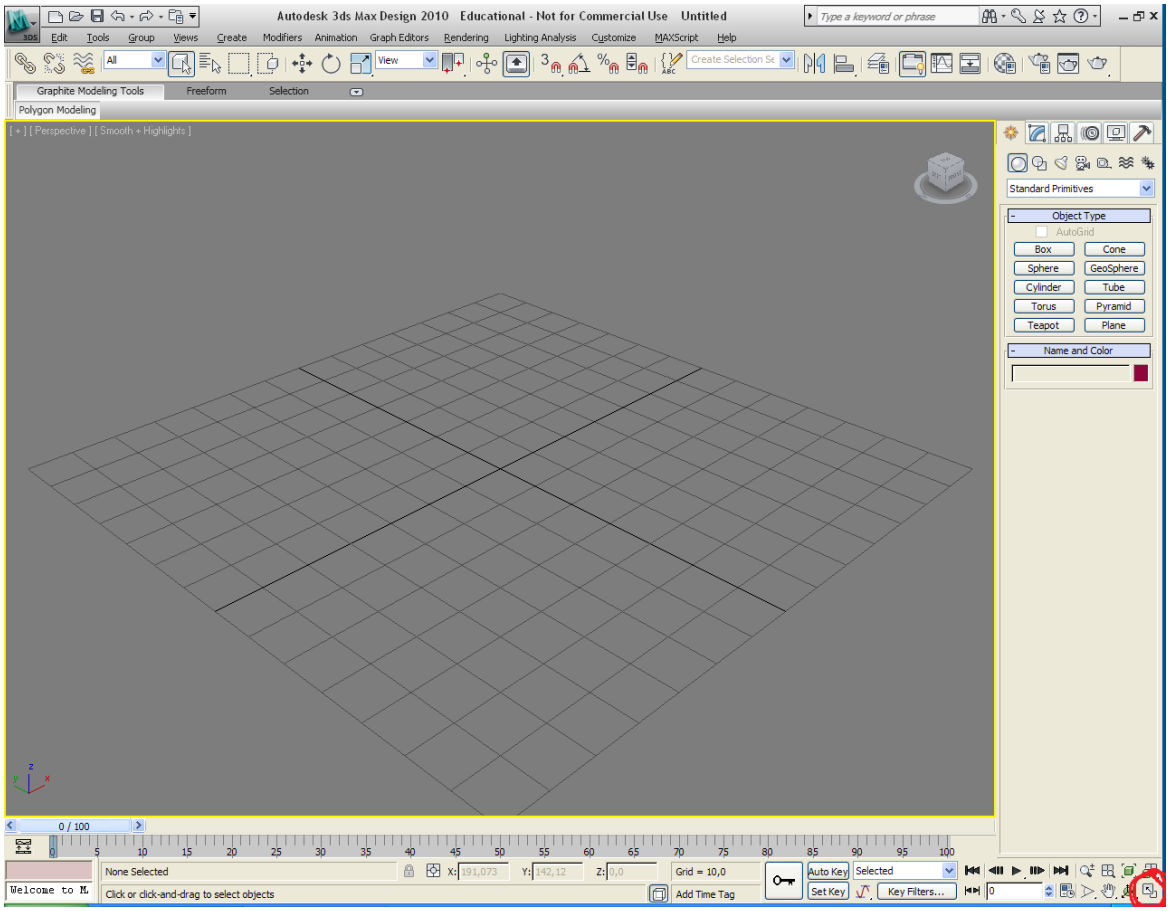
One problem you will notice is that it is rather empty; we will start by populating the island with some buildings and trees. Or whatever else you want, really. There are, of course, many other problems, which we will deal with later, but first, consider the following sections, which will help you create a more interesting scene.

When using the modified island you will create in the next section, loading the objects may take a long time. Make sure you run the code in Release mode, unless you are debugging the code.

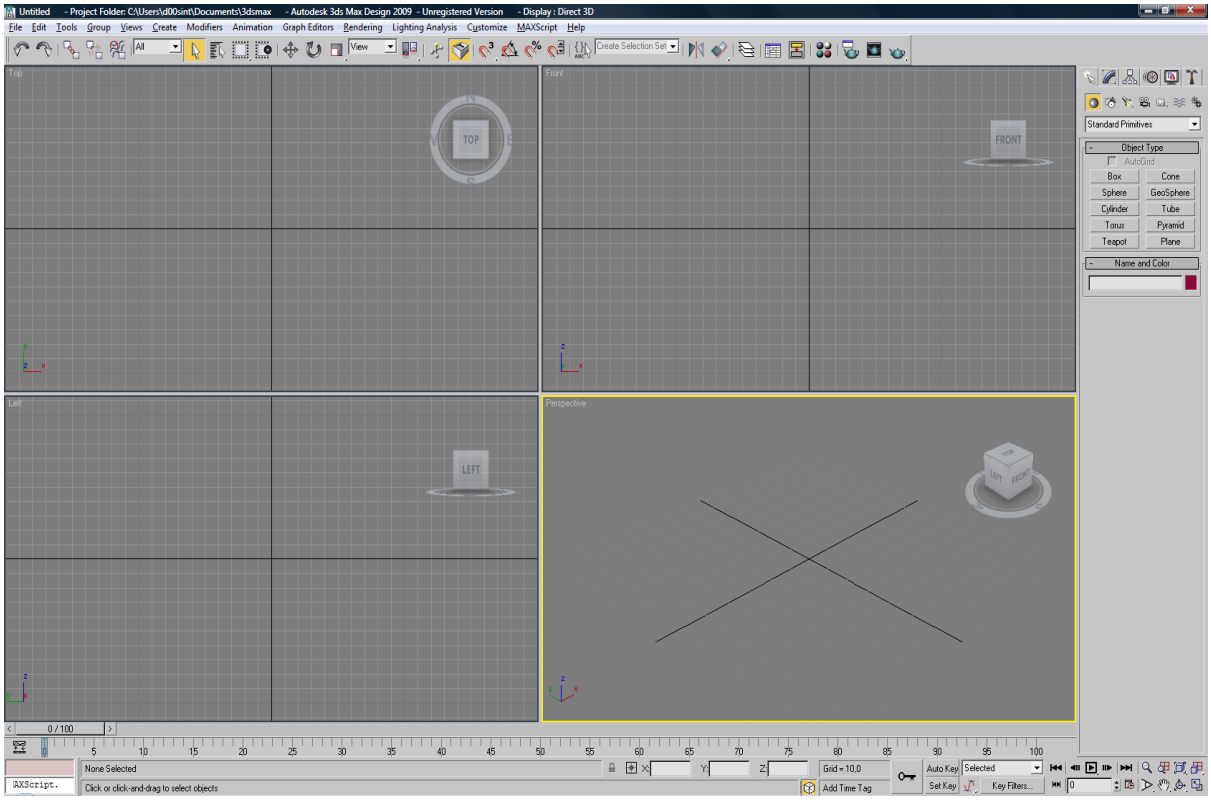
3D Studio MAX Design

3D Studio MAX Design is really the same program as 3D Studio MAX (a program heavily used in the game-industry), but with presets more suited for architects and engineers. In this part of the lab, we will take a look at the very basics of the program and then modify the island model that you will load into your program. Start 3D Studio MAX (from the start menu or the icon on your desktop). Loading the program will take a while. If the “educational use only” dialog appears, just press continue and if a dialog suggesting that you watch some introductory movies appears, click close (or watch the videos, but that is not part of the lab).

The program will look like this when you start it:



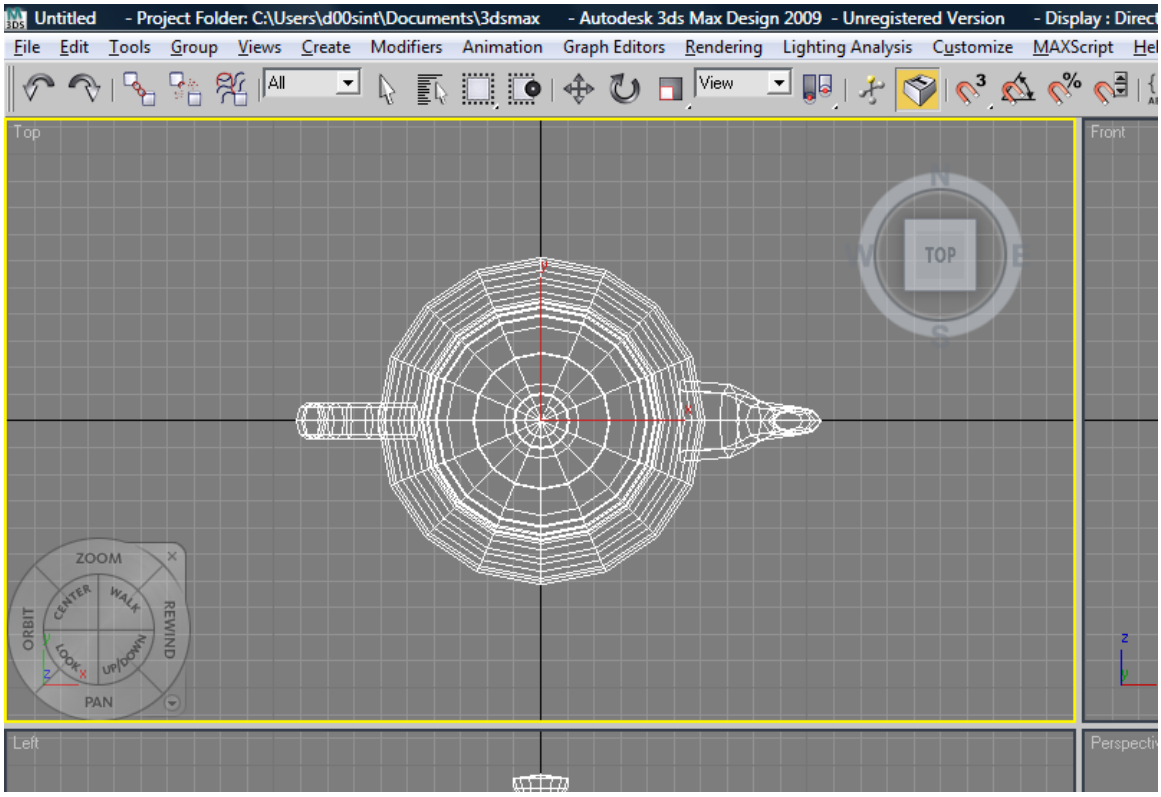
Click the button in the lower right corner to get to a more complete view:



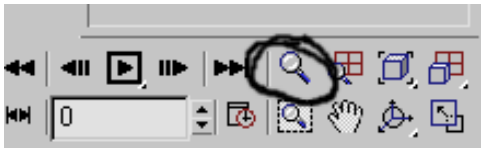
Note that the screen is divided into four windows. These are different views of your scene, and the viewing direction is written in the upper left corner (for example, the selected view with the yellow frame in the image above is the *perspective* view).

Now we will create some geometry. On the right side of the screen you see the “create geometry” tab. The “geometry” button is selected. In the “Object Type” pane, you can see buttons for creating different standard primitives (Cube, Cone, Sphere and so on). Click the button labeled “Teapot”.

Now, click the middle of the “Top” viewport and drag the mouse in any direction until you have a teapot taking up most of the viewport.

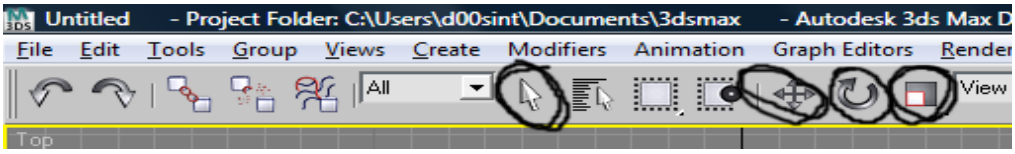


Now, you will see the teapot from different directions in the different viewports. In the lower right corner of the screen are a couple of tools for manipulating the view. Click the “zoom” tool and click and drag in the “Perspective” viewport until you can see the whole teapot.



You can use this tool in any viewport to zoom in and out. Below and to the right of the zoom tool is a hand icon. This is the “pan” tool. Select that tool and pan around a little in the “front” viewport. To the right of that tool is the “Orbit” tool. Try that in the “Perspective” viewport. This lets you view the object from any direction. Note that none of these tools move the *object*, just the cameras used for viewing the objects.

Now we want to move the teapot itself though. Look at the icons on the top toolbar:

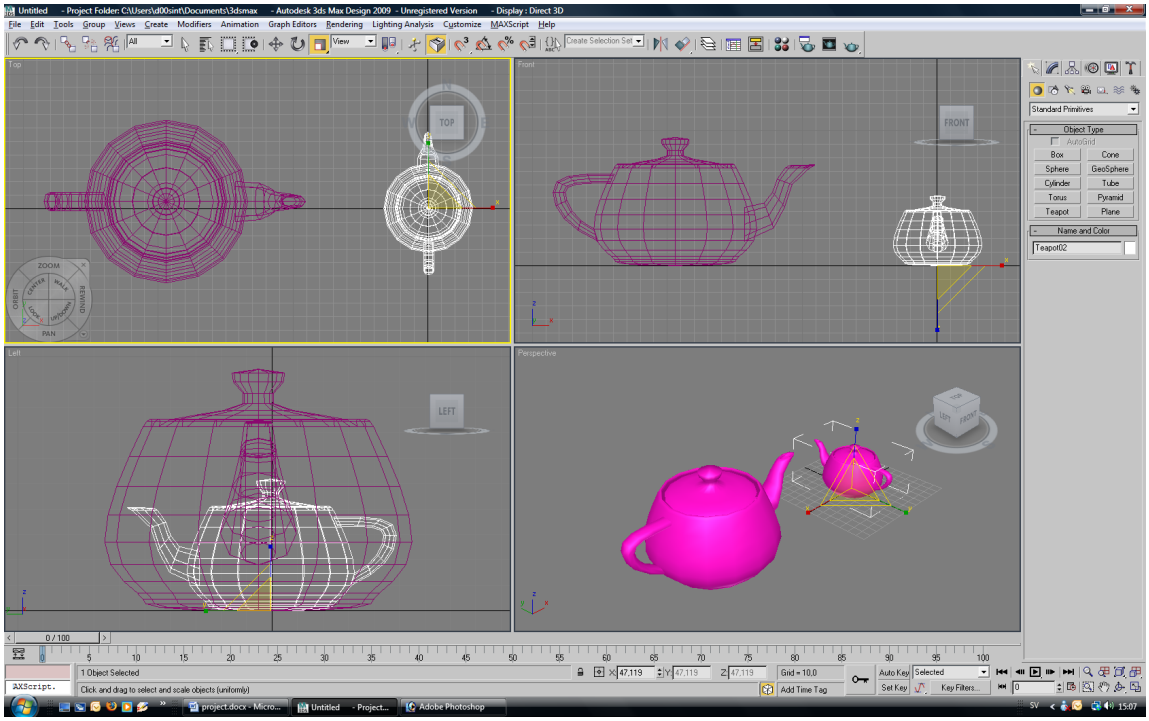


The ones marked in the image above are the ones of importance to us right now. From left to right they are the “Select”, “Select and Move”, “Select and Rotate” and “Select and Scale” tools. Click the “Move” tool and in the “Top” viewport, move the teapot to the left to make room for a new one (you can zoom out first if you need to).

Now, just like you did before, create a new teapot in the “Top” viewport of about the same size as the other one. Then click the “Select and Rotate” button and rotate the new teapot around the z (up) axis until it’s spout is facing up on the screen (in the top viewport). Finally, try the “Select and Scale” tool. Make the new about half the size of the other one.

Then click the “Zoom extents all” button:

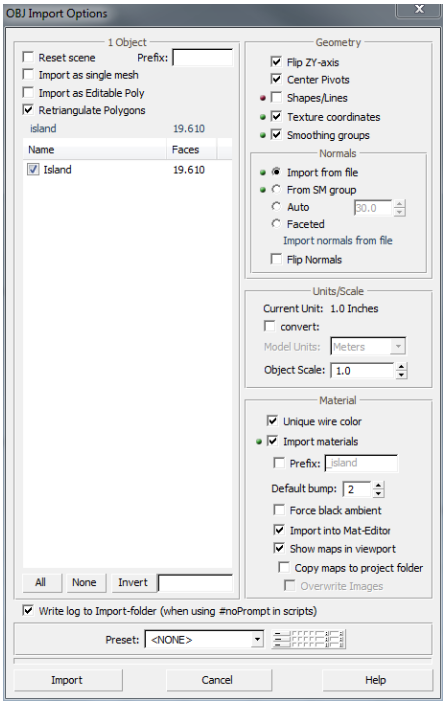
To make sure both teapots are visible in all viewports. The result should now be something like this:



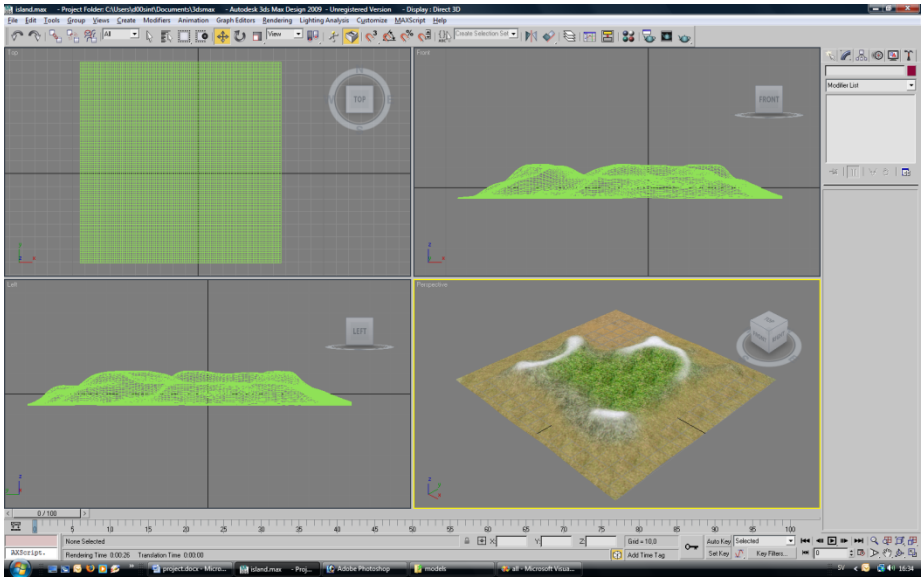
Having fun yet? If you found any of these steps confusing, play around with it some more until you get the hang of it.

Populating the island

Now we are going to load the island model you are using in the project and put some trees and houses on it to make it more interesting. First, in the “File” menu, choose *Reset* to get rid of the teapots. Then, in the “File” menu, choose “Import…”. Locate the file “island.obj” in the *scenes/* directory and open it. Make sure your import dialog looks like this:

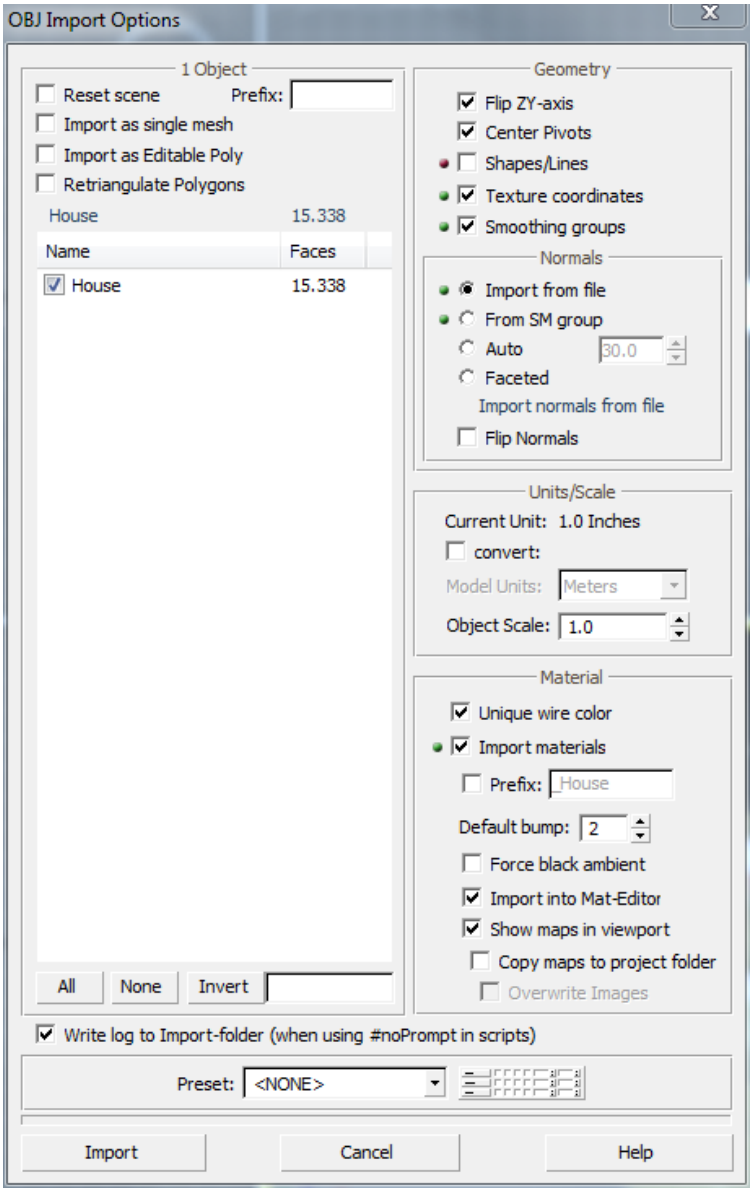


You should now have the island object showing something like this:



Explore the island using the zoom, orbit and pan tools. This might also be a good time to try the “navigation cube” showing in the top left corner of each viewport. It’s quite self-explanatory and provides a different way of navigating the scene. Not much to see on the island is there? Let’s take care of that.

Click “File->Import…” on the menu, and locate the file House.obj and open that. Make sure the dialog that shows up looks like this:



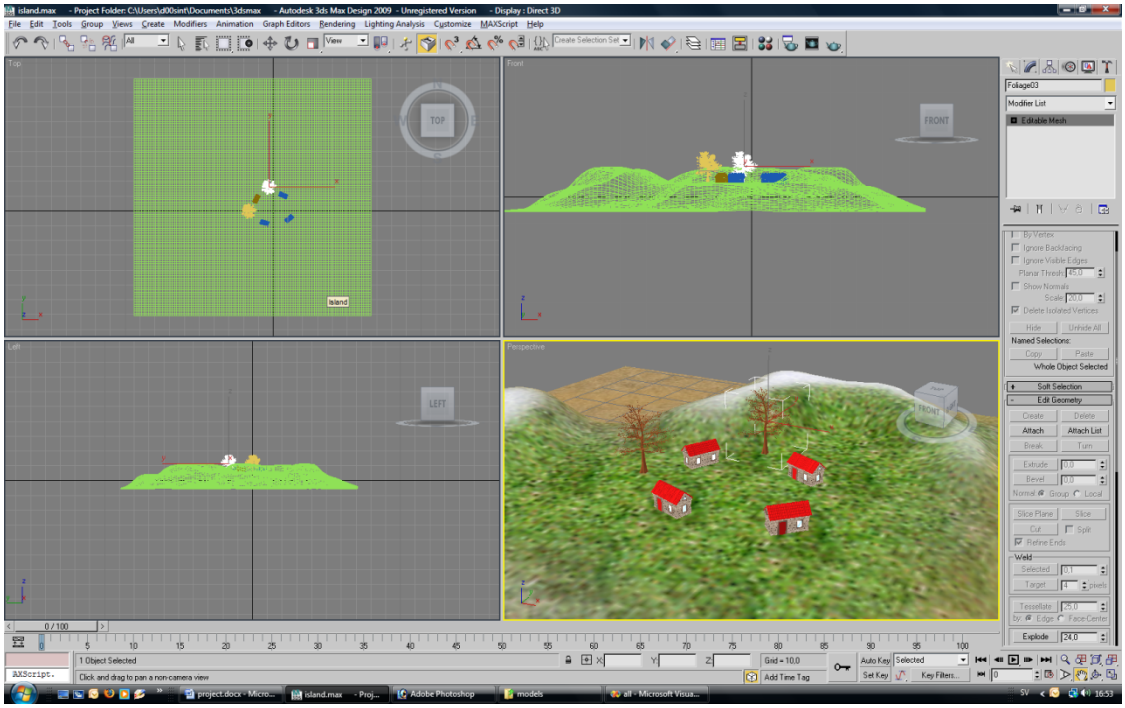
Then click import.The house will now be imported into the scene. First of all, to simplify moving the house around, select (use the Select tool) the island object then right-click on it and choose “freeze selection”. This will gray-out the island and make it impossible to select.

Now, in the left viewport, “Move” the house up along the *y* direction until it is above ground. Then move and rotate it to wherever you want it (but not right in the origo, we will put other stuff there later…). When your first house is where you want it, save your work.

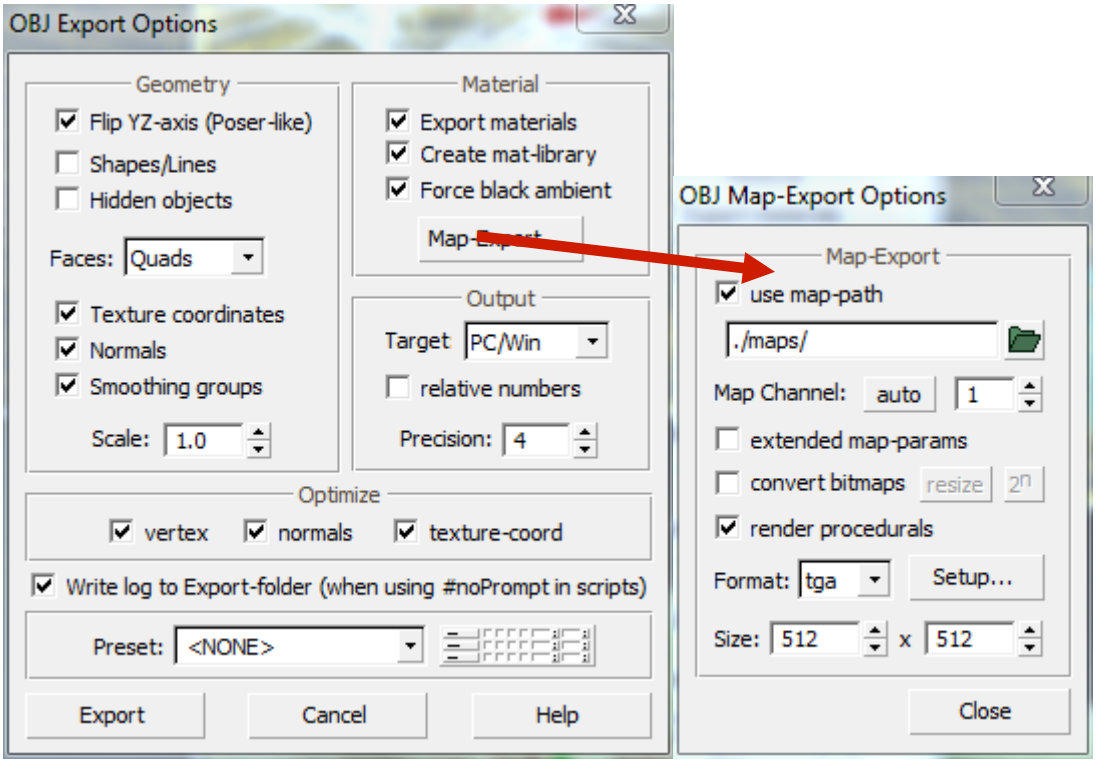
Now we will make a copy of the house. Select the “move” tool, then hold the “shift” key and in the top viewport move the house. Holding shift tells 3d studio that we want to *clone* the house. When you let go of the mouse button, a dialog will appear that asks you what the new house object shall be called and whether it should be a copy or an “instance”. Make sure you have “copy” selected and name the house whatever you want. Now put the new house wherever you want it and save your work again.

For a tad more diversity, choose “File->Import…” again and import the Tree.obj file. Just as with the house, move the tree into a good position and clone it if you want to. You can put as many houses or trees on the island as you please, but the more you put there, the slower the model will be to load in you program, so don’t go crazy.

When you’re done, right click in any viewport and select “Unfreeze all”. Your scene might now look something like this:



Save your work again. Now it's time to export the scene into the OBJ format, that your own program can read. Choose "File->Export…". Export the file in the "*scenes/*" directory and call it island2.obj. Make sure the dialog that comes up is filled in like this:



Then press export. Great, you're done.

Making it not suck...

We have an island that is no longer barren, but it still looks fairly awful. But, by some enormous stroke of fortune, you have just learned several techniques that can be used to improve matters! In the following sections you will add shading, shadow maps and a cube map for reflections.

Shading

There is already a light source defined in the simulation, the position is given by the global variable `lightPosition`, it is updated, in the `idle` function, to make the light orbit around the world, in a sun-like fashion. This means we will have a dynamic light environment. The light position is already uploaded to the shaders, where it is a uniform variable called `lightPosition`, this is what you will be working with to create the shading.

Start by implementing the lighting and shading as explained in Lab 4 (just stick to the lighting part for now, you'll be adding the environment cube map later). Remember to make sure all the positions and directions involved in the calculations are in the same reference frame (i.e. *space*, such as model or view space).

The application only provides two light intensities and colors, ambient and diffuse, use the diffuse for both specular and diffuse lighting. For the material there are a number of properties computed at the start of the shader: `diffuse`, `ambient`, `specular` and `emissive`. Note how they take into account if a texture is available.

Done? The difference in looks is nothing short of enormous. Even a simple light model accounts for a large portion of the realism in a scene. But…

Before **After** **After, at night**

Shadow Map

The scene still lacks something; it is hard to know where objects are on the island, or if they are floating around above it. We simply must add shadows! This will give us the cue that shows where the buildings are in relation to the ground. Also note that the car is lit from below (at night) as the ground does not block the light.

You already have working shadow maps in the 6th OpenGL tutorial. Add a shadow map to the application, and build the light view from the light position, looking at the origin (the island is centered on the origin). The shadow perspective projection should have a field of view that just fits the island; this should be verified by setting the shadow map border color to all zero.

In the fragment shader, let the visibility retrieved from the shadow map affect the *directional* components of the light model. That is, diffuse and specular light contributions.

Remember that you have to add all the required steps to render the shadow map, but for your convenience we have added a function `drawShadowCasters`, that should contain the code to draw all objects that can cast shadows. This function is called again to render those objects in the visible scene.

When done it should now look a lot like this, enjoy the moving shadows cast from the orbiting sun:

Cube mapping

Still, we are not really happy - the car could be shinier! After all, our specular light model only handles glossy reflection of the incoming light approximatively. This makes the finish look a lot like plastic.

Now, we'd rather it looked like car paint should. To do this we will add a cube map. Dig into Tutorial 4 again and bring out the cube map goodness. Don't forget the Fresnel term!

Add the cube map to the project and fragment shader, and then add the Fresnel computations. The R0 value is already defined. The final reflectiveness should be scaled by the per-object uniform `object_reflectiveness`, which we use to make sure the car is the only thing reflective.

Hopefully we will now have a car that looks a lot like the below picture. The effect in a still shot like this is fairly subtle, but is a lot more apparent when moving the camera.

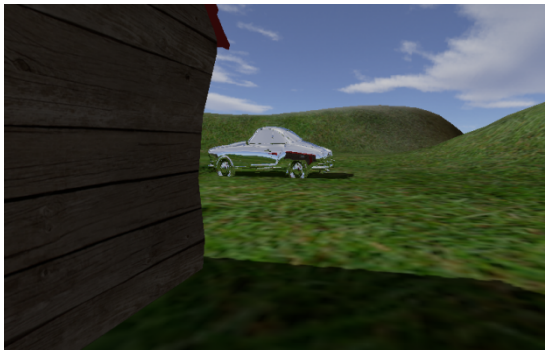
Before

After

YOUR FINAL TASK: ADD AT LEAST ONE OPTIONAL EFFECT FROM BELOW.

Optional: Dynamic Cube Maps

Add 6 frame buffer objects that each has a different cube face attached (see the OpenGL spec about `glFramebufferTexture2D`), then render to these using views that center on the car and look in the 6 axis directions. For testing purposes, you can make the car much more reflective, and also do away with the Fresnel term.



Remember to not render the car into the cube map, as it will show up as weird bits here and there. When done correctly the car should look like chrome and you should be able to see the environment change as the day goes.

Optional: A procedural sun

In the solution executable you can see the sun move across the sky. This is a simple procedural shader effect. It is created by taking the dot product of the normalized direction to the light, and normalized position of the fragment. When this is close to 1.0 we know the fragment is in the direction of the light. If this is applied to the sky dome, we get a basis for creating this effect.

It is probably a good idea to create a new shader specifically for the sky dome. This is usually a good idea anyway as it does not need to be lit or translated as does other geometry.

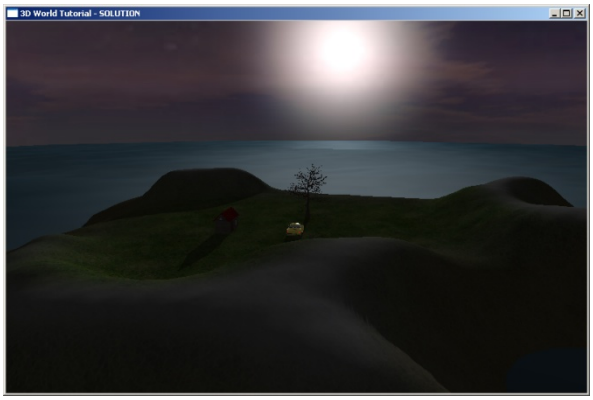
Optional: Bloom

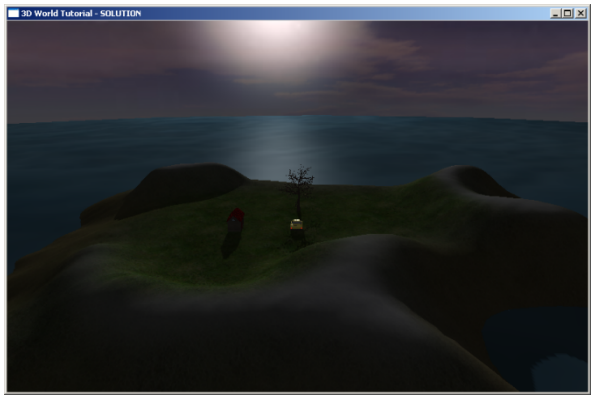
In Lab 5 you (optionally) implemented the bloom effect. Port this to the project, and hopefully, you'll get some glowing parts, like, for instance, the headlights. Play around with the cutoff level (maybe adjust it according to "daytime") to control when and where the blooming occurs.

Optional: More fun in the sun

We have implemented the *Phong* specular shading model. This function creates round specular highlights. On many objects, especially flat surfaces like the water, this is not very realistic. In the course you have been taught another model, *Blinn–Phong* commonly called the half-vector formula. This is based on micro facet theory and produces a more realistic specular highlight.

Implement this and notice how the highlight on the water changes as the sun gets lower. In the pictures below the left uses Phong and the right, Blinn–Phong.





Optional: Other Post-processing Effects?

Several post-processing effects are presented in Lab 5. You can implement a combination of those (perhaps not the magic mushrooms, though), or find/invent an entirely different post processing effect.