Project 4 – Ashikhmin-Shirley BRDF

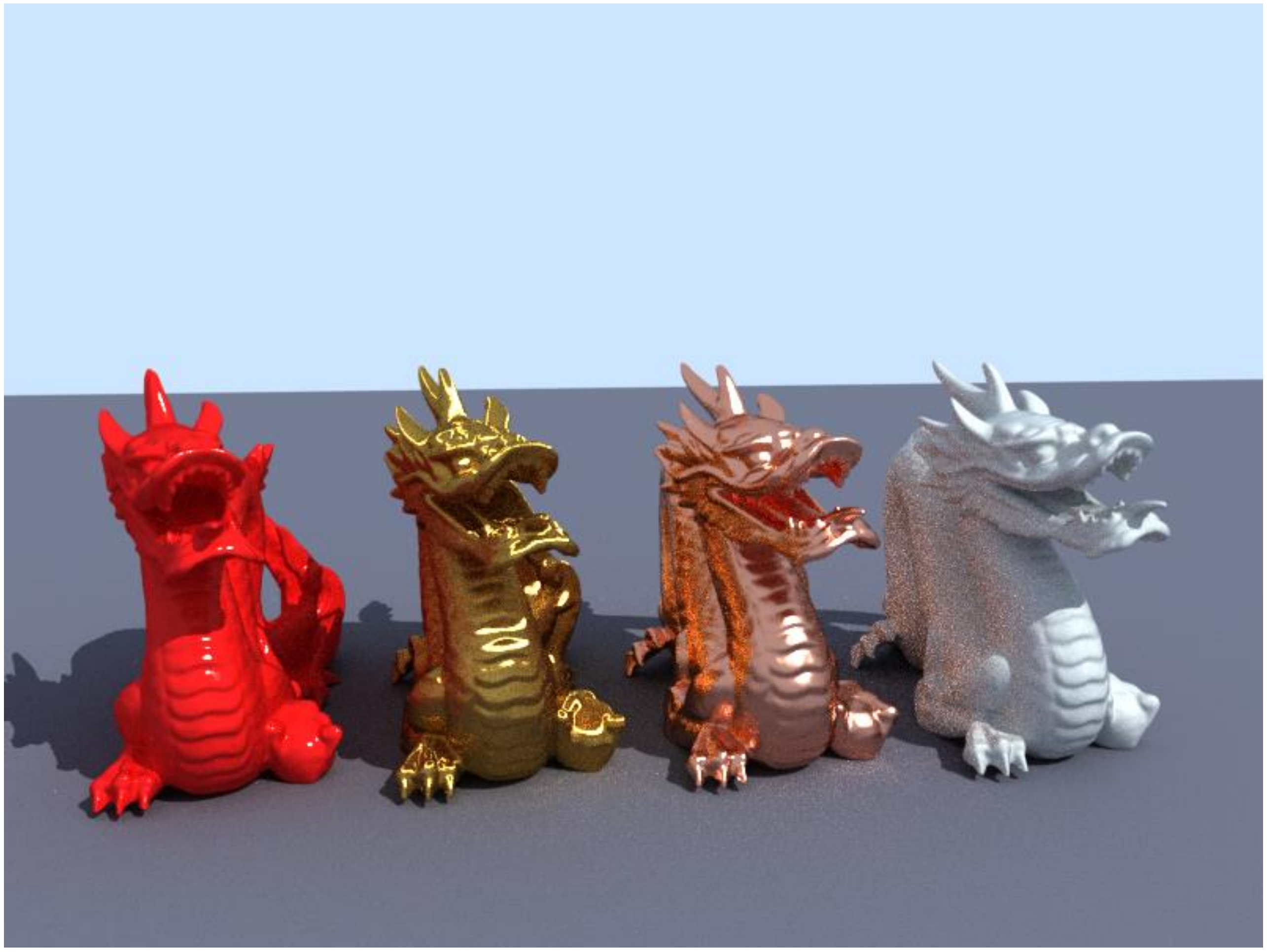
CSE 168: Rendering Algorithms, Spring 2017

Description

Implement the Ashikhmin-Shirley BRDF model. It should implement the appropriate ComputeReflectance() and GenerateSample() functions.

Sample Image

Project 4 should generate the following image (this used 10x10 rays per pixel, with up to 10 bounces). Notice the subtle gold and copper caustics caused by focused light reflected off of the metal surfaces onto the ground. Note that some parts of the paper are open to interpretation, so images may vary slightly:



Project 4 Function

Project 4 should be able to be run with the following sample code (or something very similar):

```
void project4() {
	// Create scene
	Scene scn;
	scn.SetSkyColor(Color(0.8f, 0.9f, 1.0f));

	// Materials
	const int nummtls=4;
	AshikhminMaterial mtl[nummtls];

	// Diffuse
	mtl[0].SetSpecularLevel(0.0f);
	mtl[0].SetDiffuseLevel(1.0f);
	mtl[0].SetDiffuseColor(Color(0.7f,0.7f,0.7f));

	// Roughened copper
	mtl[1].SetDiffuseLevel(0.0f);
	mtl[1].SetSpecularLevel(1.0f);
	mtl[1].SetSpecularColor(Color(0.9f,0.6f,0.5f));
	mtl[1].SetRoughness(100.0f,100.0f);

	// Anisotropic gold
	mtl[2].SetDiffuseLevel(0.0f);
	mtl[2].SetSpecularLevel(1.0f);
	mtl[2].SetSpecularColor(Color(0.95f,0.7f,0.3f));
	mtl[2].SetRoughness(1.0f,1000.0f);

	// Red plastic
	mtl[3].SetDiffuseColor(Color(1.0f,0.1f,0.1f));
	mtl[3].SetDiffuseLevel(0.8f);
	mtl[3].SetSpecularLevel(0.2f);
	mtl[3].SetSpecularColor(Color(1.0f,1.0f,1.0f));
	mtl[3].SetRoughness(1000.0f,1000.0f);

	// Load dragon mesh
	MeshObject dragon;
	dragon.LoadPLY("dragon.ply");

	// Create box tree
	BoxTreeObject tree;
	tree.Construct(dragon);

	// Create dragon instances
	glm::mat4 mtx;
	for(int i=0;i<nummtls;i++) {
		InstanceObject *inst=new InstanceObject(tree);
		mtx[3]=glm::vec4(0.0f,0.0f,-0.1f*float(i),1.0f);
		inst->SetMatrix(mtx);
		inst->SetMaterial(&mtl[i]);
		scn.AddObject(*inst);
	}

	// Create ground
	LambertMaterial lambert;
	lambert.SetDiffuseColor(Color(0.3f,0.3f,0.35f));
```

```
        MeshObject ground;
        ground.MakeBox(2.0f,0.11f,2.0f,&lambert);
        scn.AddObject(ground);

        // Create lights
        DirectLight sunlgt;
        sunlgt.SetBaseColor(Color(1.0f, 1.0f, 0.9f));
        sunlgt.SetIntensity(1.0f);
        sunlgt.SetDirection(glm::vec3 (2.0f, -3.0f, -2.0f));
        scn.AddLight(sunlgt);

        // Create camera
        Camera cam;
        cam.LookAt(glm::vec3(-0.5f,0.25f,-0.2f), glm::vec3(0.0f,0.15f,-0.15f));
        cam.SetFOV(40.0f);
        cam.SetAspect(1.33f);
        cam.SetResolution(800,600);
        cam.SetSuperSample(10,10);
        cam.SetJitter(true);
        cam.SetShirley(true);

        // Render image
        cam.Render(scn);
        cam.SaveBitmap("project3.bmp");
}
```

Grading

This project is worth 15 points:

-	Ashikhman BRDF Evaluation	7
-	Ashikhman BRDF Sampling	8
-	Total	15

Notes

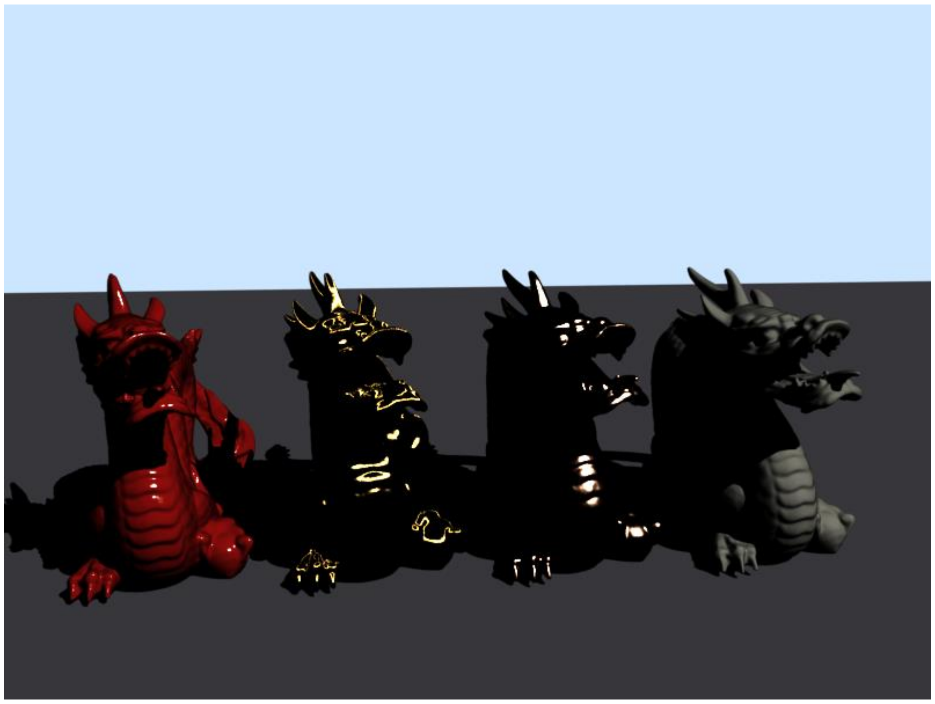
Ashikhmin BRDF

Details on the Ashikhmin BRDF can be found at:

http://www.cs.utah.edu/~michael/brdfs/jgtbrdf.pdf

Forward BRDF Evaluation

To get things started, I suggest first implementing the forward evaluation of the BRDF and ignoring the sampling function and recursive ray reflections. If you render the image without any reflections and just computing the direct lighting on the BRDF, it will look like this:



This shows how the BRDF looks when lit from a single directional light only. The rest of the color in the final image comes from reflection rays hitting other surfaces in the environment.

Tangent Vectors

You will need to support tangent vectors in the Intersection class. I suggest adding a vec3 TangentU,TangentV; to the Intersection class. Computing correct tangents requires proper texture coordinates, and the sample models do not have any. Therefore, I suggest adding something like the following in Triangle::Intersect() after computing the normal:

```
hit.TangentU=glm::cross(glm::vec3(0,1,0),hit.Normal);
if(glm::length(hit.TangentU)<0.0001)
       hit.TangentU=glm::cross(glm::vec3(1,0,0),hit.Normal);
hit.TangentU=glm::normalize(hit.TangentU);
hit.TangentV=glm::cross(hit.Normal,hit.TangentU);
```

Noise

The rendering will have a lot of noise if you don't take many samples. While testing, you will probably want to use far fewer samples per pixel to keep the render speed fast. This is how the image would look with only 2x2 samples: