List of assignments for Computer Graphics II

(last update 1. 3. 2024)

Completion of the course will be by awarding a graded credit for the project submitted. Individual assignments from the list below will be assessed. The maximum number of points that can be obtained for each assignment is given. It is expected that the source code and images of the results will be presented and commented on during the personal presentation of the project.

The project is expected to be demonstrated during the month of May.

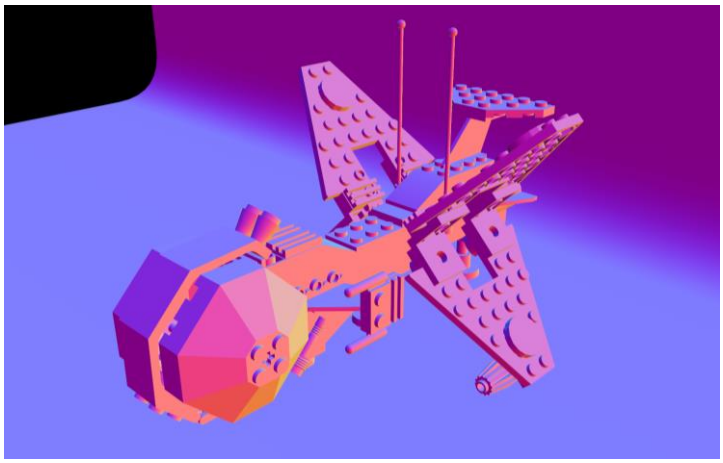
1. Task 15 points

Camera implementation according to lectures. Ideally, create an Update method in the Camera class that will calculate all necessary matrices (*V*, *P*) based on the current camera parameters. The individual matrices and their combinations (*MVP* for vertex to Clip Space transformation, *MV* for vertex to Eye Space transformation, *MVn* for normal to Eye Space transformation, etc.) can then be passed (via a pointer to the first element - data method) in the render loop as needed using the SetMatrix4x4 function (glutils.cpp) to the current shader. At the same time, ensure that the camera moves appropriately in the render loop.

```
void SetMatrix4x4( const GLuint program, const GLfloat * data, const char * matrix_name )
{
	const GLint location = glGetUniformLocation( program, matrix_name );

	if ( location == -1 )
	{
		printf( "Matrix '%s' not found in active shader.\n", matrix_name );
	}
	else
	{
		glUniformMatrix4fv( location, 1, GL_TRUE, data );
	}
}
```

2. Task (see tutorial_1 and tutorial_7 in template) 15 points

Loading the model from the OBJ file including materials from MTL. The OBJ and MTL loader is part of the template. Just follow the examples from tutorial_1 and tutorial_7 (in tutorials.cpp) to implement filling VAO and VBO objects with the loaded data. The functionality can be tested using, for example, a simple normal shader. The result should look like the attached image.



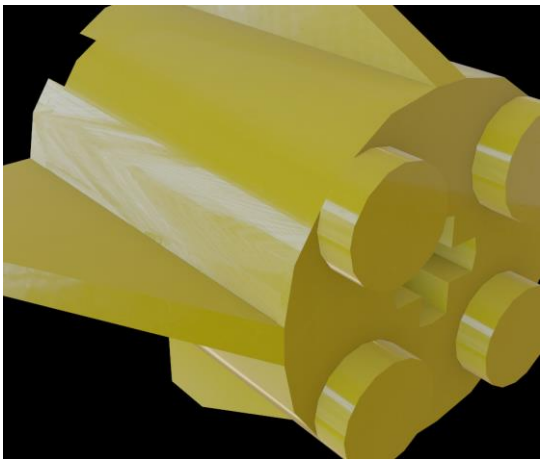
In terms of code organization, I recommend creating a Rasterizer class that will contain the implementation of this and other tasks. The structure of the implemented methods can be as follows:

```
int test( const std::string & file_name )
{
	Rasterizer rasterizer( 640, 480, deg2rad( 45.0f ), Vector3( 200, 300, 400 ),
Vector3( 0, 0, 30 ), 1.0f, 1000.0f );
	rasterizer.InitDevice(); // initialize OpenGL context
	rasterizer.InitPrograms(); // initialize VS a FS shaders (program)
	rasterizer.LoadScene( file_name ); // load scene geometry
	rasterizer.InitBuffers(); // initialize VAO and VBO
	rasterizer.InitMaterials( 0 ); // initialize SSBO for materials
	// initialization of precomputed maps for Cook-Torrance GGX shader follows
	rasterizer.InitIrradianceMap( "../../../data/lebombo_irradiance_map.exr" );
	rasterizer.InitPrefilteredEnvMap( {
		"../../../../pg2/data/lebombo_prefiltered_env_map_001_2048.exr",
		"../../../../pg2/data/lebombo_prefiltered_env_map_010_1024.exr",
		"../../../../pg2/data/lebombo_prefiltered_env_map_100_512.exr",
		"../../../../pg2/data/lebombo_prefiltered_env_map_250_256.exr",
		"../../../../pg2/data/lebombo_prefiltered_env_map_500_128.exr",
		"../../../../pg2/data/lebombo_prefiltered_env_map_750_64.exr",
		"../../../../pg2/data/lebombo_prefiltered_env_map_999_32.exr" } );
	rasterizer.InitGGXIntegrMap( "../../../data/brdf_integration_map_ct_ggx.png" );
	rasterizer.MainLoop(); // rendering loop

	return EXIT_SUCCESS;
}
```

3. Task 15 points (+ 15 bonus points)

PBR shader implementation according to lectures. All precomputed maps are available on the course website including a simple model with PBR material. If the textures of each material will be solved using bindless textures, you will get 5 extra points. If you implement the precomputation of the individual maps yourself, you will earn an additional 10 points. Thus, a total of 15 points can be earned.

Using a simple cube model [1], the result could look like this.

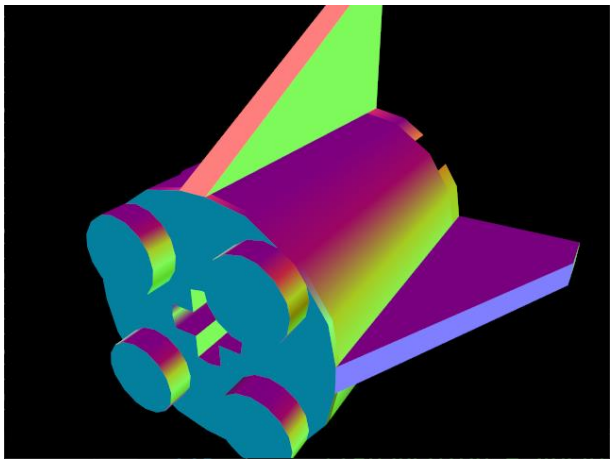


4. Task 15 points (this task is already implemented in the provided template)

To solve this problem, it is necessary (for example) to extend the constructor of the Triangle class to calculate tangent vectors in each vertex of the triangle.

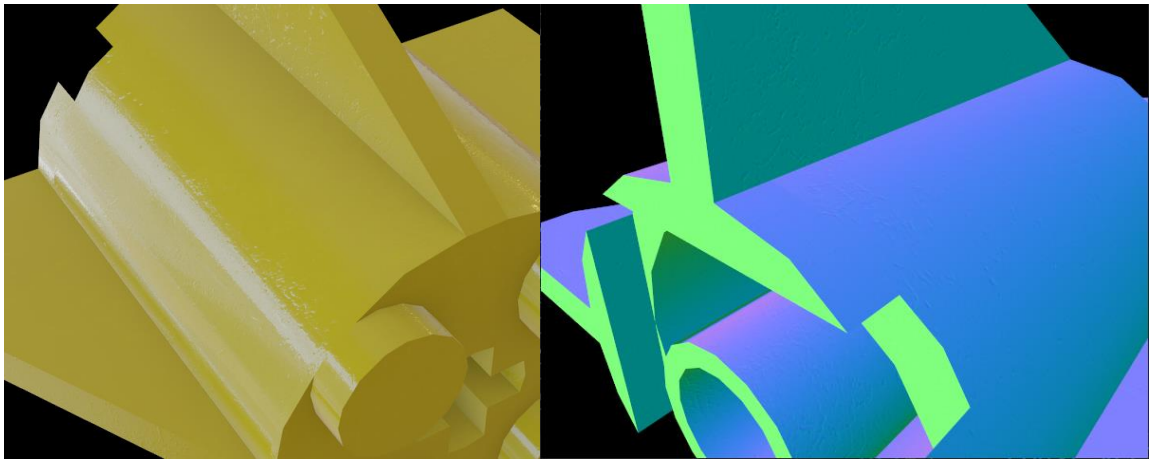
```
Triangle::Triangle( const Vertex & v0, const Vertex & v1, const Vertex & v2, Surface *
surface )
{
	vertices_[0] = v0;
	vertices_[1] = v1;
	vertices_[2] = v2;

	// TODO TBN
	const Vector3 e1 = v1.position - v0.position;
	const Vector3 e2 = v2.position - v0.position;
	// t = ?, b = ?
	vertices_[0].tangent = Tangent( t, b, v0.normal );
	vertices_[1].tangent = Tangent( t, b, v1.normal );
	vertices_[2].tangent = Tangent( t, b, v2.normal );
}

// auxiliary function for calculating the tangent vector of a given vertex of a triangle
// input is tangent, bitangent and normal before GS orthogonalization process
// output is the tangent after the GS orthogonalization process
Vector3 Tangent( const Vector3 & t, const Vector3 & b, const Vector3 & n )
{
	// TODO Gram-Schmidt orthogonalization
	return tc;
}
```

Using a simple cube model and plotting the calculated tangents using a normal shader, the result could look like this:



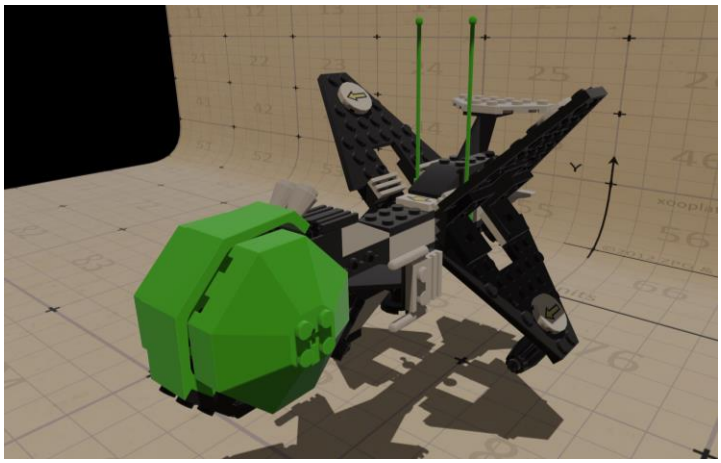
5. Task 15 points

Implementation of support for normal map using TBN according to lectures. Note the reflections affected by irregularities on the cube surface simulated using the normal map. In the previous case, the surface appeared completely smooth. Verify the correctness first using a normal shader, using which the surface wrinkling should also be apparent.



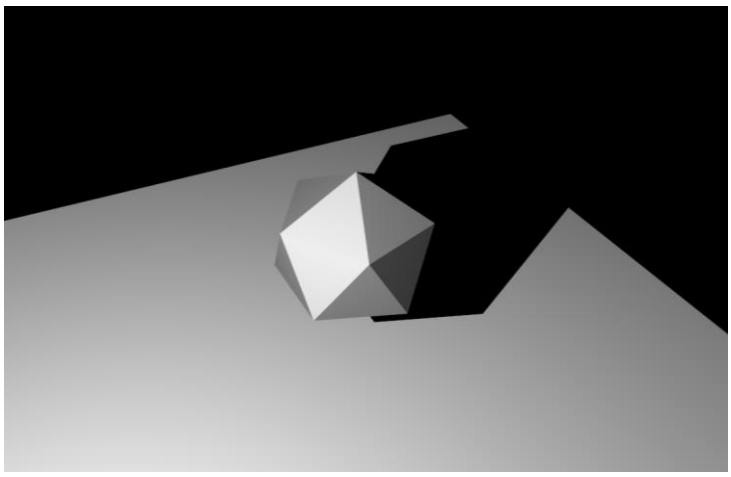
6. Task 15 points

Implementing shadows using shadow mapping. One light source of spot light (central projection) or directional light (parallel projection) is sufficient. You do not need to implement omni-directional point sources (omni light), where you would need to create 6 spot light sources and the resulting shadow map would be represented as a cube map. To verify the correctness of the implementation, you can use the model [2] and ensure that the light source moves (e.g., along a circular trajectory).



7. Task 20 points

Shadow implementation using stencil buffer. One spot light source is sufficient (center or rectangular projection). It is also sufficient if the implementation works with a simplified model [3].



8. Task 5 points

Background rendering using a spherical environmental (HDR or LDR) map. This task can be accomplished by loading a separate sphere model [4] and projecting "back-facing" surfaces to infinity. Due to the spherical mapping of coordinates at each vertex of the sphere model, ensuring the rendering of the mapped environmental texture is quite straightforward. It is sufficient to append/modify the map_Kd attribute to the associated MTL file with the set file name with the desired background texture [5].

References

[1] http://mrl.cs.vsb.cz/people/fabian/pg2/piece_02.7z
[2] http://mrl.cs.vsb.cz/people/fabian/pg2/enclosure.7z
[3] http://mrl.cs.vsb.cz/people/fabian/pg2/shadow_volume_test.7z
[4] http://mrl.cs.vsb.cz/people/fabian/pg1/sphere.zip
[5] https://polyhaven.com/a/lebombo