CS4345 General Purpose Computation on GPU (2016/2017 Semester 1)

Programming Assignment 1

Release Date: 1 September 2016, Thursday
Submission Deadline: 18 September 2016, Sunday, 11:59 PM

LEARNING OBJECTIVES

Writing basic shaders to use stored textures and to perform procedural texture mapping. After completing the programming assignment, you should have

- learned how to express positions and directions in eye space,
- learned how to convert positions and directions between eye space and local tangent space,
- understood varying, attribute, and uniform variables,
- learned how to access built-in attributes and uniform variables,
- learned how to do per-fragment lighting computation,
- learned how to access a 2D texture map, and
- learned how to access a cubemap.

TASK

You are to write a vertex shader and a fragment shader to perform **procedural bump mapping** and **reflection mapping**. The following images show sample results that your shaders should produce when being used on a **square**, a **cube** and a **cylindrical tube**:

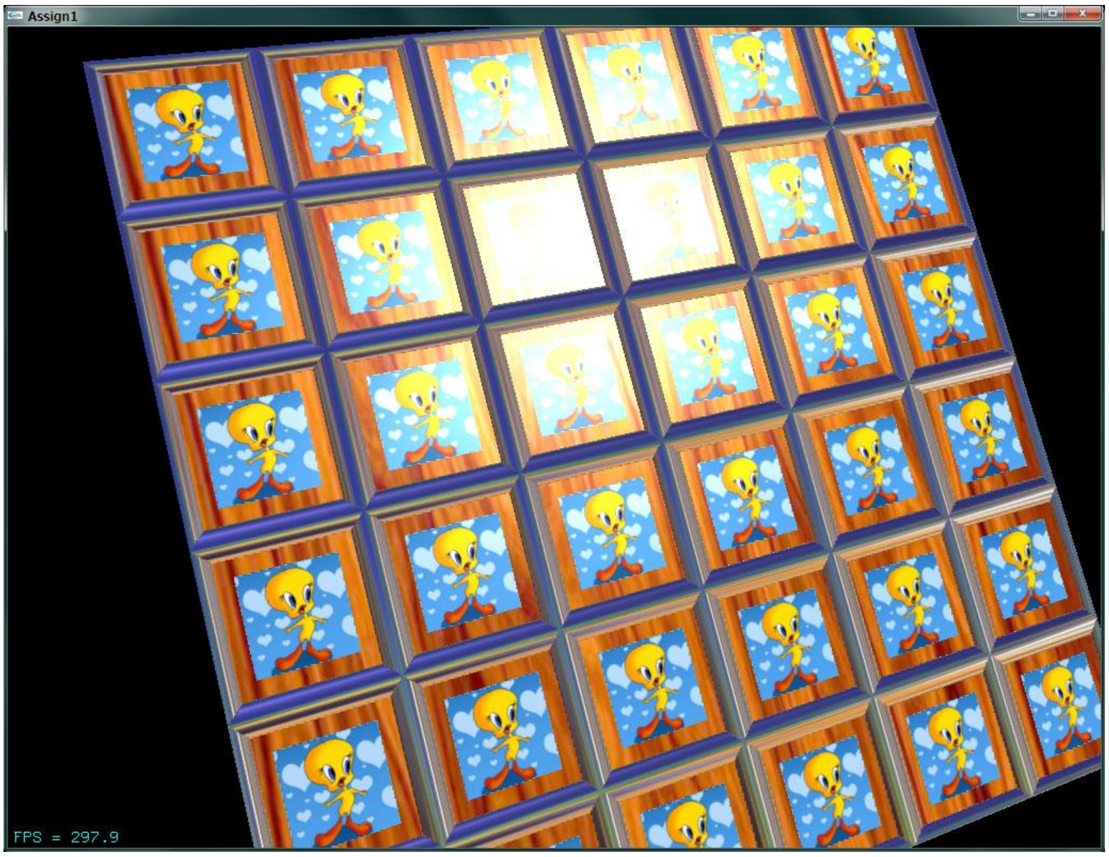

Figure 1

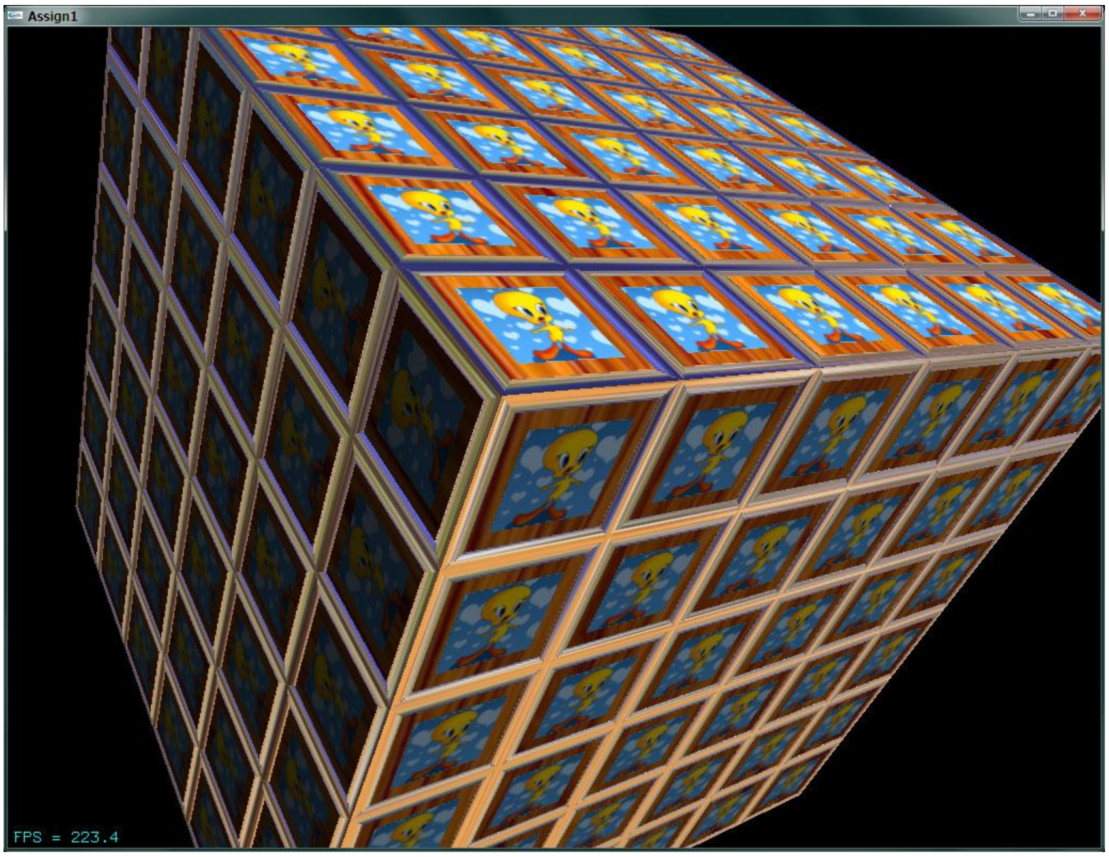

Figure 2

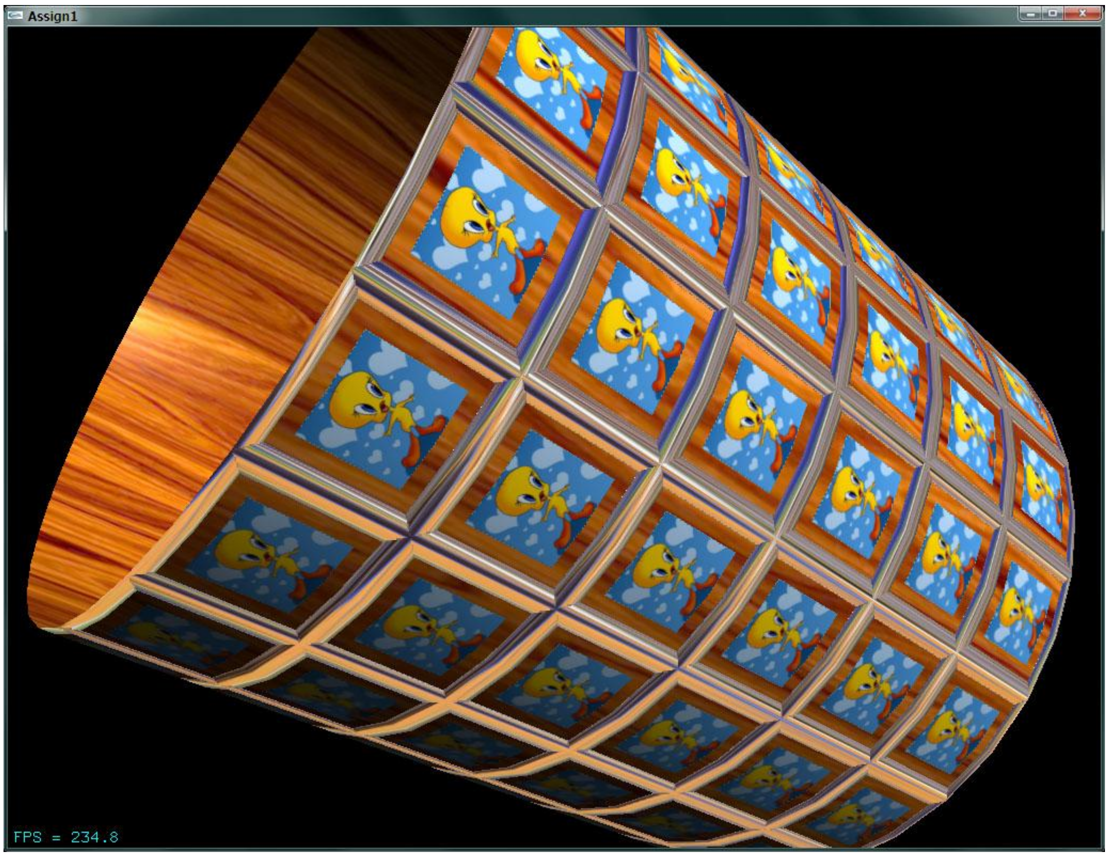

Figure 3

You are provided with a pair of **incomplete GLSL shaders**, and your job is to complete them according to the requirements.

Please download the ZIP file **cs4345_assign1_2016_todo.zip** from the **Assignment** folder in the IVLE Workbin. You can try the Win32 executable program **assign1.exe** found in the same ZIP file. Please read the instructions shown in the console window to learn how to operate the program. The program does not produce correct rendering right now since it is using the incomplete shaders.

The incomplete shaders are in the files **assign1.vert** and **assign1.frag**. All the necessary **uniform**, **attribute**, and **varying** variables have already been declared. **You must not add new uniform, attribute and varying variables**. You can add new functions in your shaders. Note that you should adhere to the **variable naming convention** where the prefix "ec" is used to indicate that the entity is expressed in the eye space, and the prefix "tan" to indicate tangent space.

A C++ OpenGL application program **assign1.cpp** and other source files have been provided in the same ZIP file for you to **study carefully**. A Visual Studio 2008 solution **assign1.sln** is also provided for you to rebuild the executable program. The application program loads the shader source files **assign1.vert** and **assign1.frag**, and use them in the rendering. It also provides the values for the **attribute** variables and **uniform** variables to the shaders. In this assignment, **you are not required and must not change any of the C/C++ source files**. Of course, you can modify them for your own experiments, such as changing the input values of the uniform variables, but you need not submit the modified C/C++ source files. Your shaders should produce results same as those in the above sample images when evoked by the original C/C++ program.

Your shaders should produce the effect of a **wooden surface** tiled with a grid of **tube-shaped mirrors** and **square sticker images**. The followings describe the required effect in more details.

Wooden Surface

The wooden surface is texture-mapped with a wood image. It is visible at places not covered by the reflective mirrors and the sticker images. The surface must be lit using **per-pixel Phong lighting computation**. This Phong lighting computation code is exactly the same as that used in the Phong shading example that I showed you in class, except that the **scene ambient (`gl_FrontLightModelProduct.sceneColor`), ambient and diffuse components of the lighting result must now be modulated (multiplied) by the color read from the wood texture map, and the result is then added to the specular component to get the final fragment color.**

Note that the "**backside**" of each surface has no mirror or sticker image, and must undergo the same lighting computation as the "frontside". The following shows the rendering of the "backside" of the square:

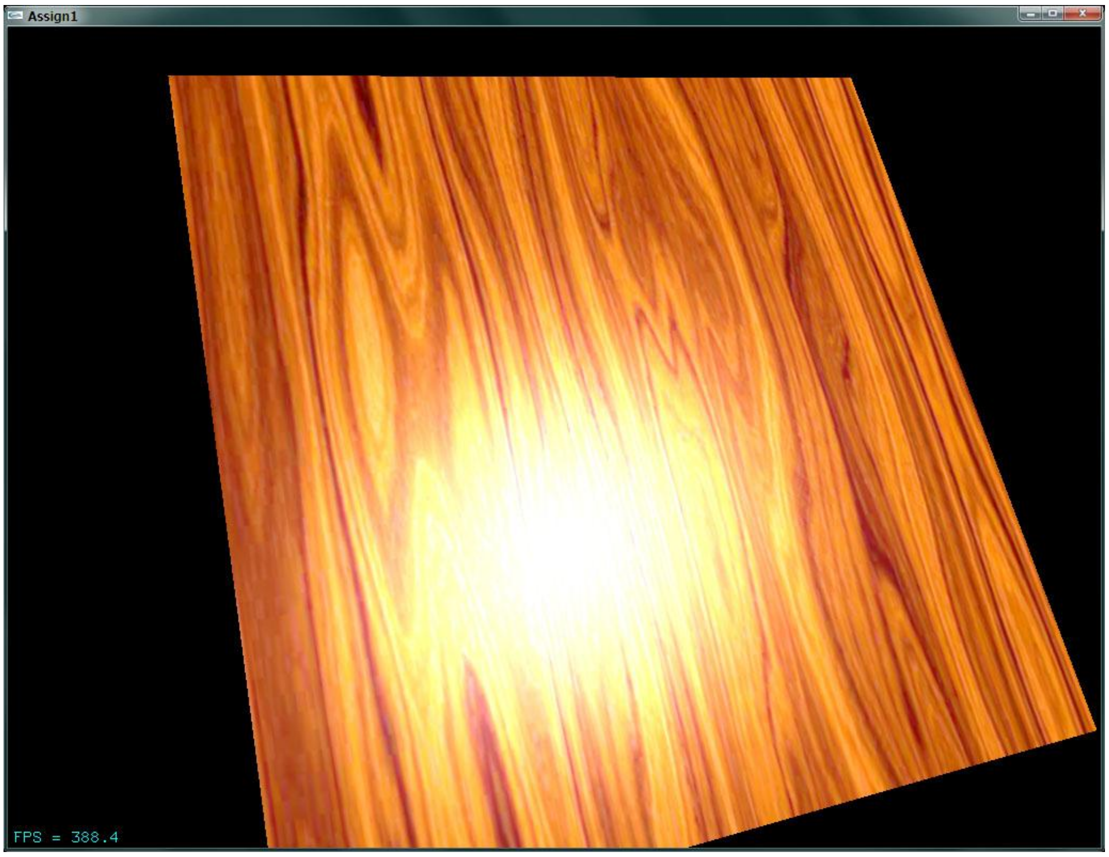

Figure 4

Tube-Shaped Mirrors

Each tube-shaped mirror *appears* to protrude from the surface (see Figure 5). The surface is divided into a 2D grid of tiles, where the number of tiles across the surface (in a single dimension) is determined by the uniform variable “TileDensity”. Each tile is surrounded by tube-shaped mirrors, each running along a boundary edge of the tile.

The **cross-section of each tube is a semi-circle**, whose radius is specified by the uniform variable “TubeRadius”. This radius is relative to the tile width and ranges between 0.0 and 0.5.

You need to modify the fragment shader, and use the technique of **bump mapping** to produce the “bumps” of the tube-shaped mirrors. On the bumps, you are to add **perfect mirror reflections of the environment**. The **environment map** has already been set up by the application program as a **cubemap**. Since these are perfect mirror reflections, the color value retrieved from the cubemap should be used directly as the final fragment color. **No lighting needs to be computed**.

Note that since lighting need not be computed, it is more convenient to transform the perturbed normal vector from the fragment’s tangent space to the eye space, and compute the mirror reflection vector in the eye space directly. This is because the environment is assumed to be in the eye space, and therefore the cubemap must be accessed using a direction vector in the eye space.

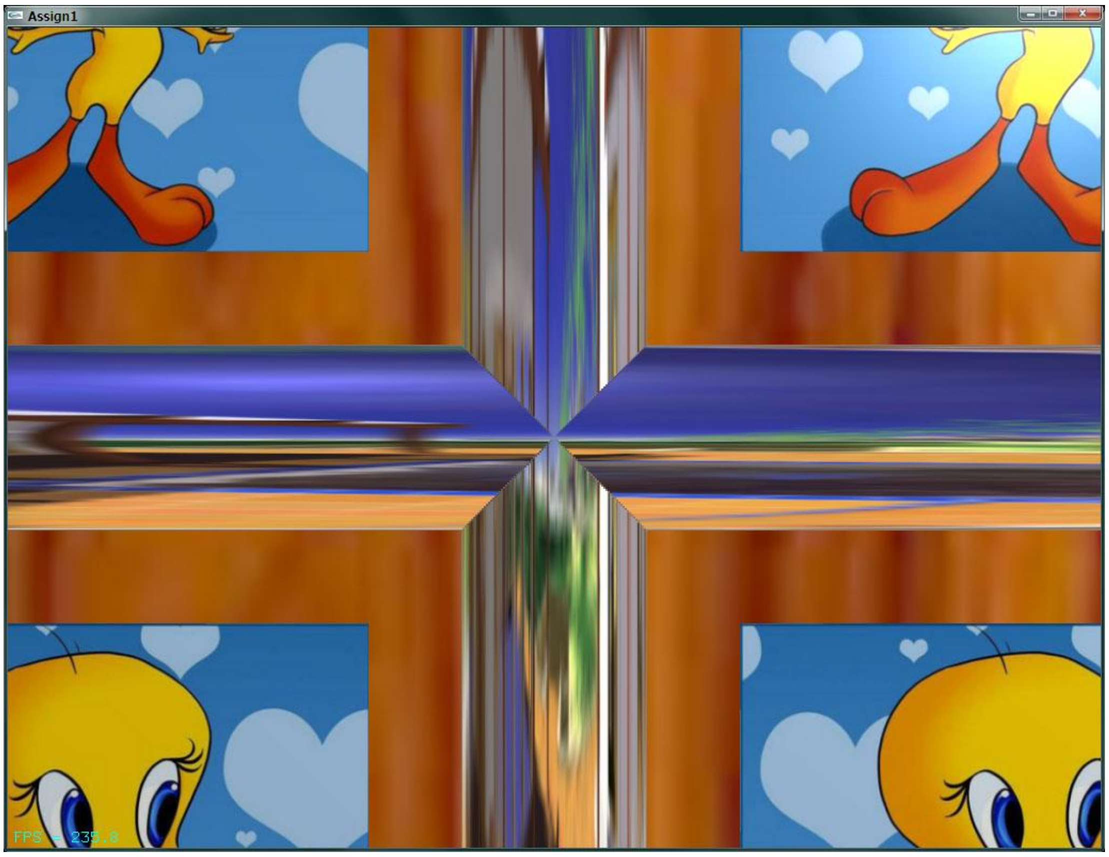

Figure 5

Square Sticker Images

At the center of every tile is a square sticker image. The sticker's width relative to the tile width is specified by the uniform variable "StickerWidth". Note that the given texture image must appear in its entirety in the sticker image (no cropping should happen).

GRADING

The maximum marks for this programming assignment is **100**, and it constitutes **7%** of your total marks for CS4345.

Program **correctness** constitutes **90 marks** while **design/style** constitutes **10 marks**. Note that if your shaders cannot be compiled and linked, you get 0 (zero) mark for program correctness.

Good coding style. Comment your code adequately, use meaningful names for functions and variables (adhere to the new variable naming convention), and indent your code properly. You must fill in your **name**, **matriculation number**, and **NUS email address** in the **header comment**.

SUBMISSION

For this assignment, you need to **submit only the two completed shader source files**:

- **assign1.vert**
- **assign1.frag**

You must put them in a ZIP file and name your ZIP file **<*matric_no.*>.zip**. For example, **A0123456X.zip**. All letters in your matric. number must be capitalized.

Submit your ZIP file to the **Assignment 1 Submission** folder in the IVLE Workbin. Before the submission deadline, you may upload your ZIP file as many times as you want to the correct folder. **We will take only your latest submission.** Once you have uploaded a new version to the folder, you **must delete the old versions**. Note that when your file is uploaded to the Workbin folder, the filename may be automatically appended with a number. This is fine, and there is no need to worry about it.

DEADLINE

Late submissions will NOT be accepted. The submission folder in the IVLE Workbin will automatically close at the deadline.

——— End of Document ———