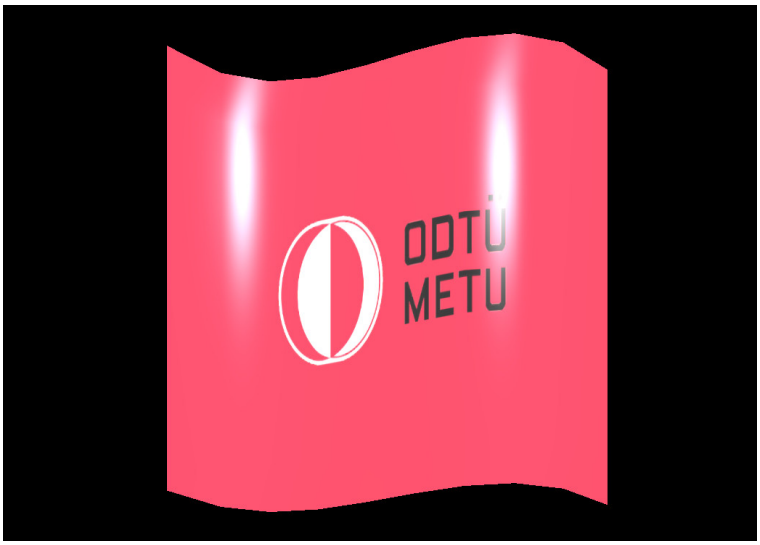
Middle East Technical University | Department of Computer Engineering

CENG 469

Computer Graphics II

Spring 2021-2022
Assignment 1
Surface Rendering in OpenGL
(v1.0)

Due date: April 10, 2022, Sunday, 23:59



1 Objectives

In this assignment, you are going to implement a smooth surface motion in OpenGL. The expected end product of this assignment is an OpenGL program which:

- Renders an animated and textured flag on the screen,
- Shows the specular shading effect of a point light on the animated flag,
- Includes user interaction capabilities with keyboard buttons.

The specifications are explained in Section 2. Note that you are free to implement your own design choices as long as you obey the requirements detailed in this assignment text. Design choices include but are not limited to:

- Flag animation characteristics and direction,
- Texture image of the flag,
- More than one point lights in the scene,
- OpenGL draw call primitive type (GL_TRIANGLES, GL_TRIANGLE_STRIP, etc).

2 Specifications

1. Bezier surface equation should be used as the flag's surface equation whose control points (CPs) will be animated at each frame.
2. The CPs should be animated continuously and smoothly (e.g. in a "back and forth" motion). There should not be any sudden jumps in the animation. Animating a CP means editing the world position of one or more of its 3D coordinates. As an example, $sin()$ or $cos()$ functions can be used for a smooth and consistent animation along all CPs.
3. From the surface equation, whose CPs are animated at the beginning of each frame, the vertices that are to be used in OpenGL rendering will be sampled as a uniform grid. Suppose the user-interactable parameter **samples** is 10 during that frame, then 10x10 vertices should be sampled that uniformly covers the surface equation. The user should be able to increase/decrease the value of this parameter using keyboard buttons during runtime as follows:
    - The parameter's initial value should be 10.
    - Tapping **W** button should increase its value by 1. There is no upper value limit.
    - Tapping **S** button should decrease its value by 1, down to a minimum value of 2.
4. Triangles should then be generated using the sampled vertices. The per-vertex normals that will be interpolated in the shaders should then be calculated for each vertex by averaging the normals of the triangles that include this vertex. Do not use the normal calculation on the slides – this is because if you do so, normals may not match the actual rendered geometry (they will match the surface of course, but this may create a mismatch with what is actually being rendered).
5. The flag can be composed of one or multiple Bezier surfaces. A user-interactable parameter **patches** is used to declare the Bezier surface count in each of vertical and horizontal directions. If it is 1, whole flag surface is generated using just one Bezier surface equation. If it is increased to 2, then 4 Bezier surface equations will compose the whole flag surface (2x2, where each equation generates 1 quarter of the same-sized flag). **G1 continuity** should hold between multiple Bezier surfaces (i.e. no visible sudden edges between two neighboring Bezier surfaces). Note that this enables the flag to have a more wavy motion, as now the CP count of the overall flag surface is increased, allowing more finer animation control. The user should be able to increase/decrease the value of "patches" parameter using keyboard buttons during runtime as follows:

- The parameter's initial value should be 1.
- Tapping **E** button should increase its value by 1. There is no upper value limit.
- Tapping **D** button should decrease its value by 1, down to a minimum value of 1.

6. You are advised to use the **sampleGL.zip** file shared to you on ODTUClass course page as the template code and build your homework code on top of it. This codebase already implements the basic OpenGL setup and the scene arrangement required in this homework (a camera and a point light). It also has the vertex and fragment shaders implemented. You are advised to start early by first inspecting and understanding how sampleGL code works. You can also inspect the Matlab sample codes (**curves and surfaces.zip** file) provided to you on ODTUClass for Bezier surface equation part.

7. Any image can be used as the texture. The texture should replace the **kd** value (diffuse component) in the shading calculations of the shader (see frag2.glsl code of sampleGL.zip). Along with specular shading computations, the rendering should now show specular shading effect on the textured flag surface during the animation.

8. You can use an image reader library to read the image data from any image file type (JPG, PNG, etc.) to be used as the flag texture. As an example, you can use the header-only **stb_image.h** library as follows:

   (a) Download and include the file:

   ```
   #define STB_IMAGE_IMPLEMENTATION
   #include "stb_image.h"
   ```

   (b) Load the image:

   ```
   int width, height, nrChannels;
   unsigned char *data = stbi_load("metu_flag.jpg",
                           &width, &height, &nrChannels, 0);
   ```

9. There is no frames-per-second (FPS) constraint in the homework, but your animation should be seen continuous and smooth. As you increase the number of patches and the number of samples, you can see a slow-down in your application, which is expected.

10. **Blog Post:** You should write a blog post that shows some output visuals from your work, and explains the difficulties you experienced, and design choices you made during the implementation phase. You can also write about:

    - What kind of animation you have implemented and how.
    - How did you implement G1 continuity,
    - An analysis of **patches** vs **samples** vs **FPS** (frames-per-second) of your code.
    - Some interesting design choices you made during implementation, etc.

The deadline for the blog post is 3-days later than the deadline for the homework. You can submit your code before the deadline, and finalize the blog post during the 3-day late submission period without losing late days. You consume your late days only if you submit your "code" late. However, submitting the blog post more than three days later will incur grade deductions.

3 Regulations

1. **Programming Language:** C/C++. You also must use gcc/g++ for the compiler. Any C++ version can be used as long as the code works on Inek machines.

2. **Changing the Code Template:** We provide you a code template (sampleGL.zip on ODTUClass). You are free to edit, rename or delete any file from the given code template as long as you comply with the submission rules below. However, keep in mind that any error introduced by the changes you made is your responsibility.

3. **Additional Libraries:** GLM, GLEW, and GLFW are typical libraries that you will need (these are already included in the sampleGL source file). You should not need to use any other library except for reading images. But if you still want to use some other library, please first ask about it in the ODTUClass forum of the homework.

4. **Submission:** Submissions will be done via ODTUClass. You should submit your blog post link to "HW1 Blog Post Link Submission" assignment on ODTUClass. For code submission, create a "**tar.gz**" file named "hw1.tar.gz" that contains all your source code files, texture image, and a Makefile. The executable should be named as "**main**" and should be able to be run using the following commands (any error in these steps may cause a grade deduction):

   tar -xf hw1.tar.gz
   make
   ./main

5. **Groups:** All assignments are to be done individually.

6. **Late Submission:** You can submit your codes up to 3 days late. Each late day will be deduced from the total 7 credits for the homeworks of the semester. However, if you fail to submit even after 3 days, you will get 0 regardless of how many late credits you may have left. If you submit late and still get zero, you cannot claim back your late days. You must e-mail the assistant if you want your submission not to be evaluated (and therefore preserve your late day credits).

7. **Cheating: We have zero tolerance policy for cheating**. People involved in cheating will be punished according to the university regulations and will get 0 from the homework. You can discuss algorithmic choices, but sharing code between students is strictly forbidden. Please be aware that there are "very advanced tools" that detect if two codes are similar.

8. **Forum:** Any updates/corrections and discussions regarding the homework will be on ODTUClass. You should check it on a daily basis. You can ask your homework related questions on the forum of the homework on ODTUClass.

9. **Grading:** Your codes will be evaluated on Inek machines. We will not use automated grading, but evaluate your outputs visually. Rendering a smoothly-animated, textured, and visually visible and correct specular-shading-included flag surface with correctly implemented user-interactions will get you 100 points from the homework part. Note that you should test your code on an Inek machine before submission, as the frame rate might not be as high as your home computer if you have a powerful PC, and might cause hangs during the animation. This might require reducing/optimizing per frame calculations in the code. Note that some slow-down is expected as the number of samples/patches is increased but it should render smoothly at the initial setting. Blog posts will be graded by focusing on the quality of the content. Some sample blog posts will be shared with you.