# CS171 Assignment 2: Geometric Modeling

## Introduction

In this assignment, you are going to step into the geometric modeling section in computer graphics by generating Bézier surfaces, and other surfaces such as B-Spline/NURBS surfaces if you want. To accomplish the surface construction, you can reduce this problem into sub-problems in the lower dimension, which is the evaluation of Bézier curve. Once you are able to generate the curve, the surface is then a piece of cake.

In the following, we will give you the specifics about what you need to accomplish, as well as some related guidelines to assist your programming.

## Note

Before working on the assignment, you are recommended to read the materials on curve and surface modeling, and in particular, Bézier curves and surfaces. You can also refer to Delaunay triangulation and subdivision methods for more advanced implementations.

## Programming Requirements

- **[must]** You are required to implement the basic iterative de Casteljau Bézier vertex evaluation algorithm. [30%]
- **[must]** You are required to construct Bézier surfaces with the normal evaluation at each mesh vertex. [40%]
- **[must]** You are required to render the Bézier surfaces based on the vertex array. [10%]
- **[must]** You are required to add texture to the generated surface. [20%]
- **[optional]** You may construct the B-Spline/NURBS surfaces. [15%]
- **[optional]** You may support the interactive editing (by selection) of control points. [10%]
- **[optional]** You may stitch two or more surfaces together. [5%]
- **[optional]** You may implement the adaptive mesh construction based on the curvature estimation. [15%]

## Demonstration Requirements

In addition to programming, you will also need to demonstrate your code to TAs.

Things you should prepare:

- Explain how you implement the basic iterative de Casteljau Bézier verytex evaluation algorithm, and show the related code fragments.
- Explain how you construct the Bézier surface and evaluate the vertex normal, and show the related code fragments.
- Explain how you render the Bézier surfaces based on the vertex array, show the related code fragments, and demonstrate the result.
- Explain how you stitch multiple Bézier surface patches together, show the related code fragments, and demonstrate the result.
- For the optional part, explain your implementation and show it!

Additional Notification:

- You should try your best to present your ideas as clearly as possible.
- If you do not follow the above requirements, your score will be deduced by 10% of the entire assignment score.

## Submission

You are required to submit the following things through the GitHub repository:

- Your project codes.
- A PDF-formatted report which describes what you have done in the report folder.

Submission deadline: **22:00, Nov 12, 2024**

## Grading rules

- You can choose to do the **[optional]** item, and if you choose to do it, you will get additional scores based on the additional work you have done. But the maximum additional score will not exceed 15% of the entire score of this assignment.
- **NO CHEATING!** If found, your score for the assignment is zero. You are required to work **INDEPENDENTLY**. We fully understand that implementations could be similar somewhere, but they cannot be identical. To avoid being evaluated inappropriately, please show your understanding of your code to TAs.
- Late submission of your assignment will be subject to score deduction based on the rule on the course webpage.

## Skeleton Project/ Report Template

The skeleton program and report template will be provided once you accept the assignment link of GitHub classroom which will be given below. If you accept the assignment through the link properly, a repository that contains the skeleton project and the report template will be created under your GitHub account. Please follow the template to prepare your report.

You should complete your assignment submission to your repository through GitHub before the deadline.

# Implementation Guide

## Git Classroom

Accept the assignment in this link to start your assignment.

### 1. Environment Setup

Just like assignment 1, you will need CMake to build your code. To build the project, firstly you need to download CMake if you do not have one. Then run command

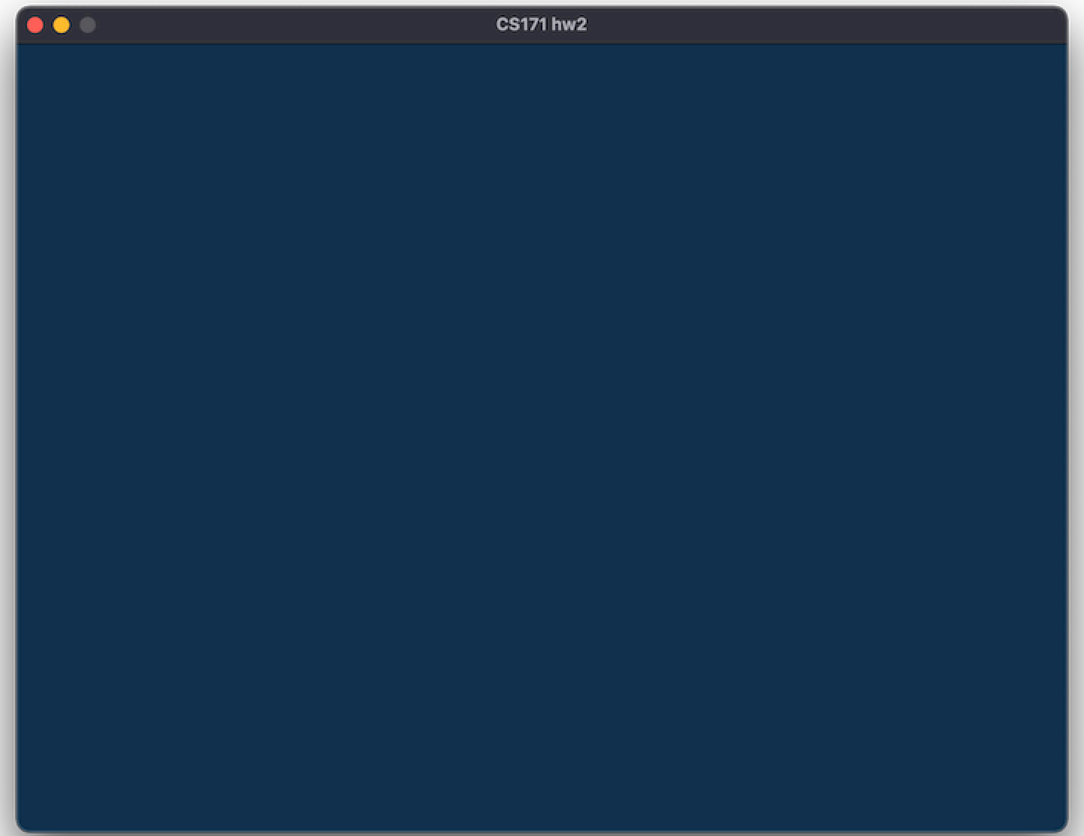
```
mkdir build
cd build
cmake ..
cmake --build .
```

These commands first make a directory whose name is "build" and then jump into it. After that, it uses CMake to configure the project and builds the project.

Besides, we recommend using Visual Studio in Windows and Visual Studio Code in Linux. Both of them can build the CMake-based projects automatically (maybe with help of some plugins). If you are an experienced developer, you can choose whatever you like.

### 2. Creating the window program using GLFW

We have already created a blank window for you to start your drawing. You should expect the following result.



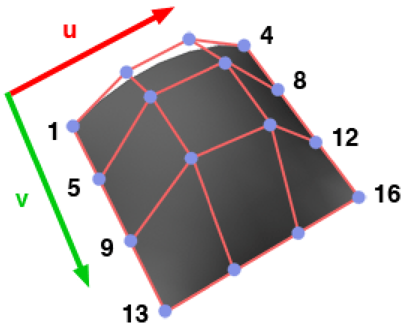

### 3. Evaluating points in Bézier curve with De Casteljau's Algorithm

Here, you are supposed to implement the function `Vertex BezierCurve::evaluate(std::vector<vec3>& control_points, float t)` in "bezier.cpp", where you need to evaluate two things of each point: the point position and the tangent. For the evaluation of positions, you can follow this site. As for the tangent, it is a by-product of the position evaluation, and thus there is no need to pay extra efforts on the tangent evaluation.

### 4. Constructing the Bézier surface

Here, you are supposed to implement the function `Vertex BezierSurface::evaluate(std::vector<std::vector<vec3>>& control_points, float u, float v)` in "bezier.cpp" for evaluating the vertices on a Bézier surface.

Since you are already able to evaluate the vertices on a Bézier curve, you can definitely utilize it to make things easier. In short, given $m \times n$ control points to evaluate the point at $(u, v)$, you can first construct $m$ intermediate Bézier curves based on the corresponding $n$ control points and evaluate m points at $v$, and then, construct another Bézier curve based on the $m$ points and evaluate the point at $u$, which is shown in the following GIF.



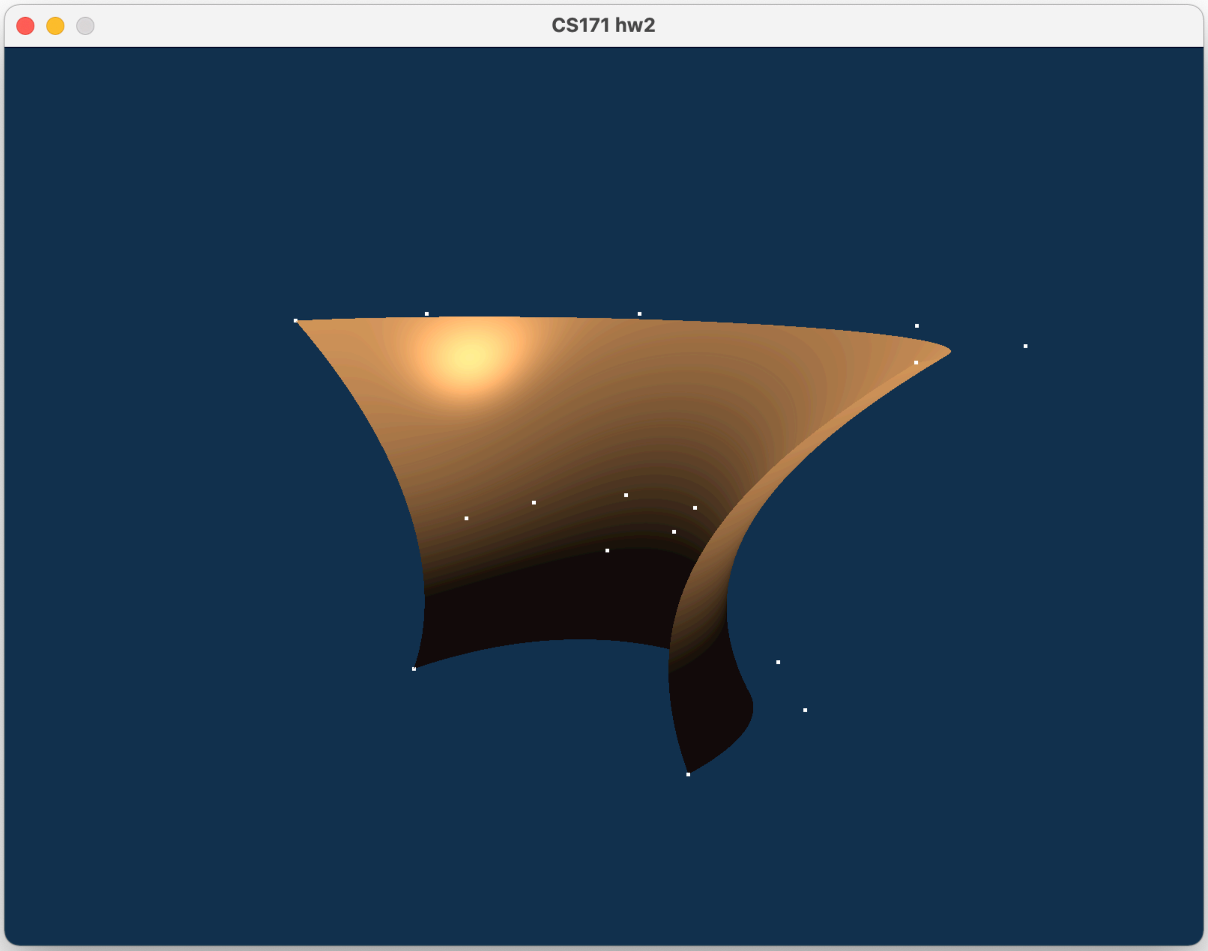

In this process, you can compute the position and the tangent along one dimension. To compute the normal, you need to apply the cross product of the tangents along two dimensions, which can be obtained along with the above process.

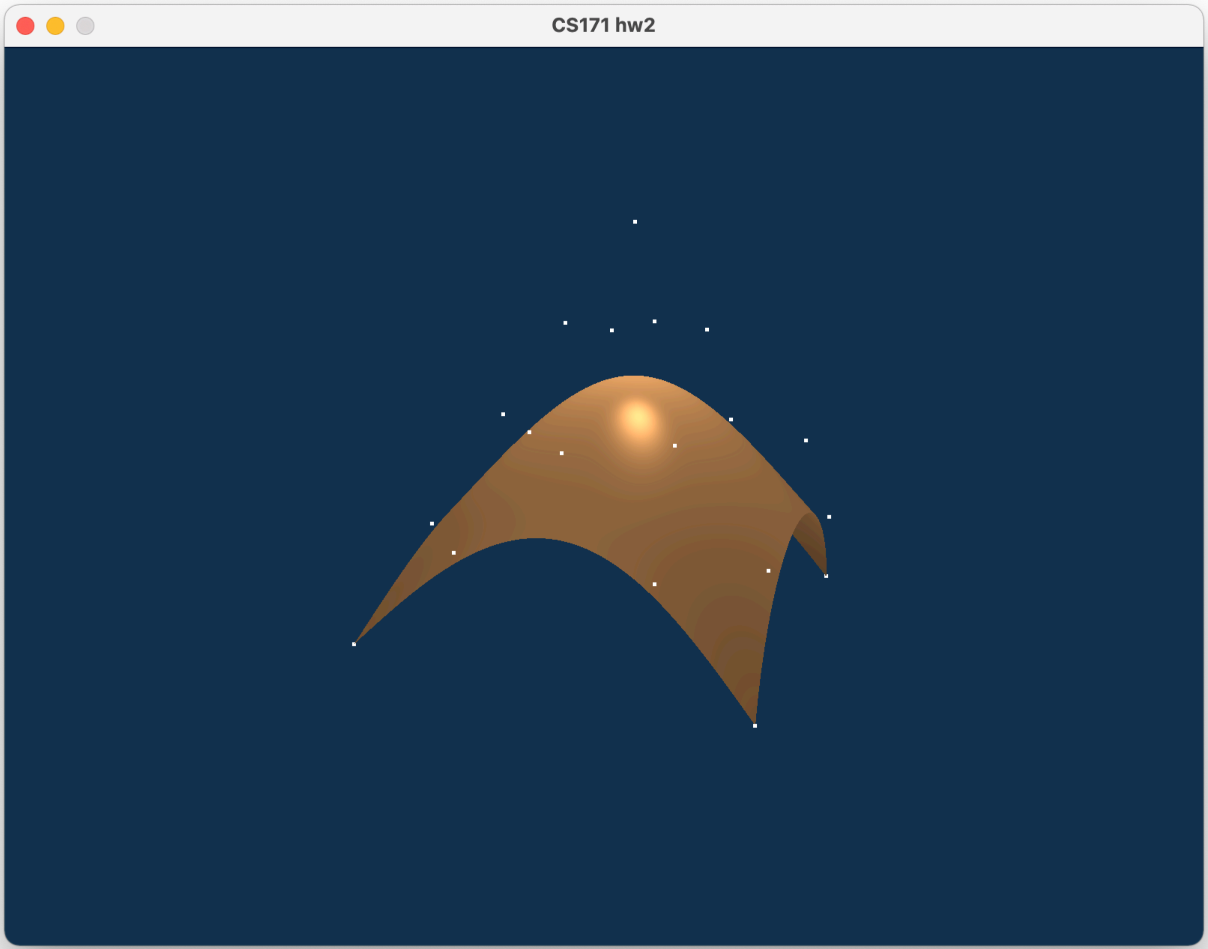
Please refer to [this tutorial]() for this part.

### 5. Rendering the Bézier surfaces

In this part, you are supposed to firstly implement the function `void Object::init()`, which initializes all the related OpenGL variables like VAO, VBO and EBO, and configure the attributes. After this, you should implement the function `void Object::drawArrays()` or `void Object::drawArrays(const Shader& shader)` to support object drawing with VAO and VBO, and the function `void Object::drawElements()` or `void Object::drawElements(const Shader& shader)` to support object drawing with VAO, VBO and EBO.

After implementing this, you can successfully render all kinds of Bézier surfaces. Here we offer some examples.

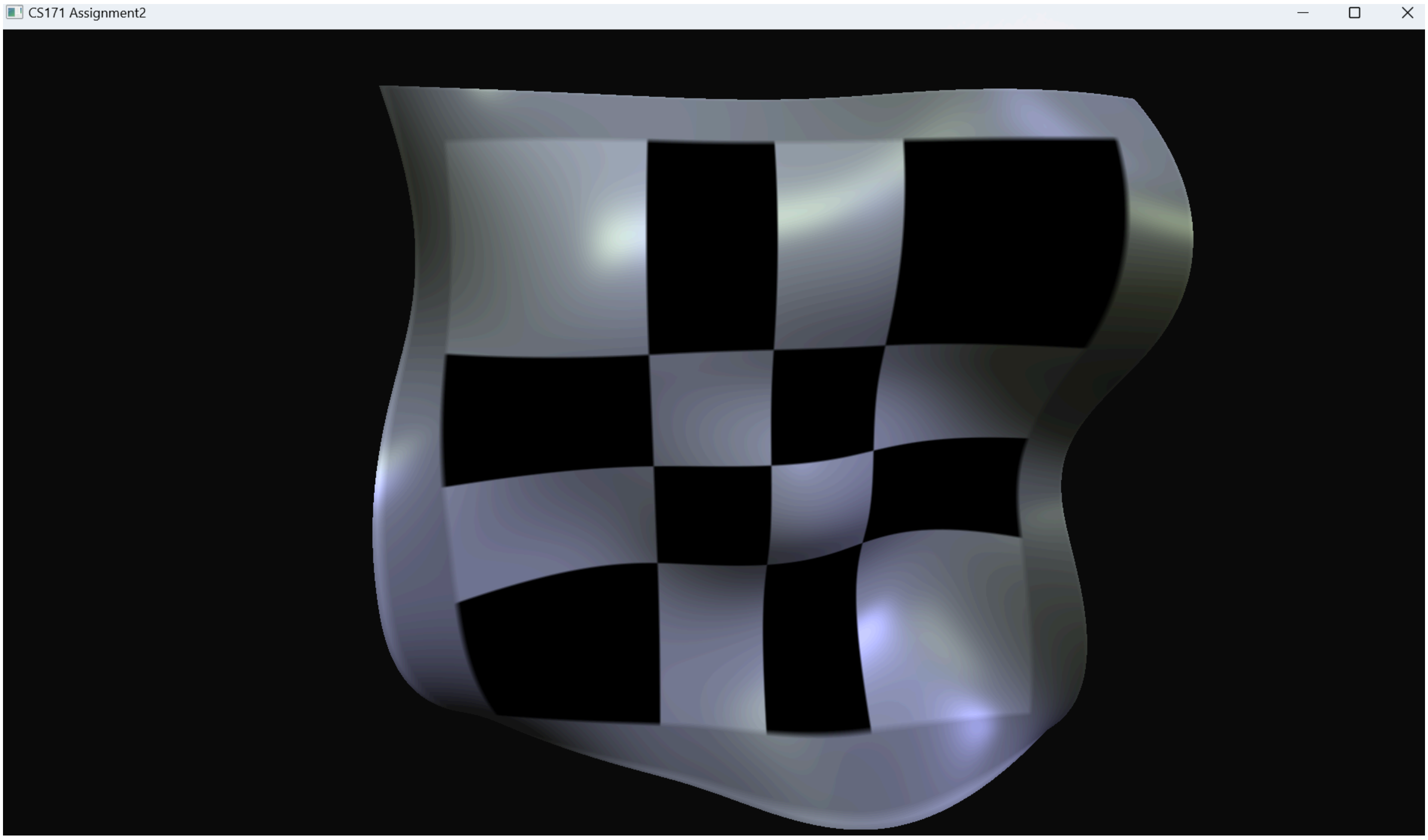

CS171 hw2




### 6. Add texture to surface

When constructing the surface triangle mesh, the UV coordinates could be assigned to each vertex easily simply by the parameter of the point. Once we have UV coordinates, we could map a texture to the surface.

To map texture to mesh in OpenGL, we need to process the UV coordinates in shaders. Besides position and normal, UV coordinates is added to each vertex in the VBO. In the vertex shader, we do not need to do any extra computation, we just need OpenGL to interpolate the UVs for us in the rasterization process, and use the interpolated UV to sample texture in fragment shader. In the fragment shader, use `texture(...)` function to sample a texture. Read this tutorial for detailed reference on texture mapping in OpenGL.




Looking forward to your exciting work!

Copyright 2024 ShanghaiTech University