Computing Science CMPT 464/764 **Spring 2021**
Instructor: Richard (Hao) Zhang **Simon Fraser University**

Assignment #1 (Total mark: 30 + 6 bonus marks): due at 23:45 on Monday, Feb. 22
Written part to be submitted as a scanned PDF on CourSys
Programming part via electronic submission on CourSys

---

*A cover page must be submitted as the **first page** of your submitted PDF. On the cover page, write and sign the following statement: "I have read and understood the policy concerning collaboration on homework and lab assignments". Without such a signed statement, your work will not be marked. A sample cover page can be found here.*

Problem 1 (18 marks + 6 bonus marks): mesh data structures, GUI, and subdivision

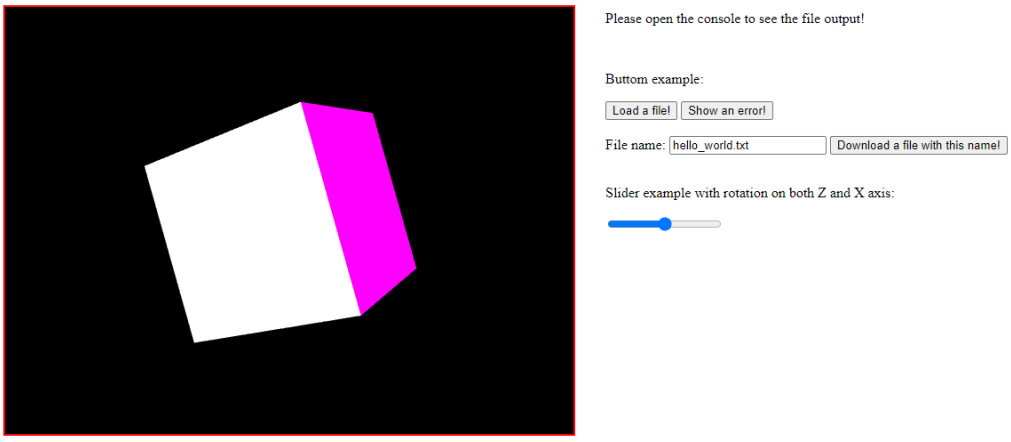
You will design and program a **graphical user interface (GUI)** for visualizing triangle meshes. The GUI may be enhanced in later assignments and/or the course project.

1. *[4 marks] Learn WebGL, if you had not during CMPT 361, and design your GUI*

   The interface you program should use **WebGL**, which has been the programming environment and graphics library of choice for most offerings of CMPT 361 at SFU in recent years. If you had no prior experience with WebGL, then you are expected to learn it mostly on your own, with some initial guidance by the TA. WebGL is a JavaScript API for rendering interactive 2D and 3D graphics within any compatible web browser without the use of plug-ins. The following screenshot shows what is generated by the demo code:

   https://coursys.sfu.ca/2021sp-cmpt-464-d1/pages/demo_code

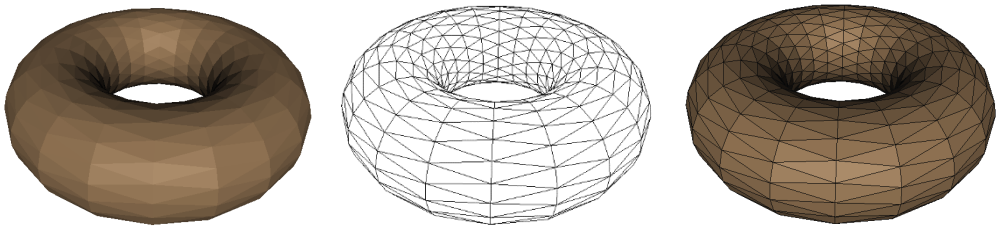
   It demonstrates how to generate a custom GUI and how to render and interact with a simple cube:



   You can run the index.html file on any modern browser directly without the need for a webserver.

You are required to design a GUI that responds to user inputs and displays a mesh model in different ways. At least the following components should be provided:

- A GUI control to allow the selection of **mesh display options**: *flat shaded, smoothly shaded, wireframe,* or *shaded with mesh edges displayed*.



- Allow rotation, zooming, and translation. Use the proper controls or events from WebGL. There are two viable options: use mouse event listeners to interact with the mesh directly or use sliders to interact indirectly (the later is demonstrated in the sample code).
- Implement the logic to input from and output to a mesh file. The sample code provides relevant snippets for loading and outputting files.

You will receive full marks for this part if your program can generate such an **interface;** there will be no need to bind the controls with the intended actions.

2. *[10 marks] OBJ parser and winged-edge mesh data structure*

Write code to input a triangle mesh given in **OBJ** format and store the connectivity and geometric information in the **winged-edge data structure.** There is a lot of information about this well-known data structure online. You can start here:

https://en.wikipedia.org/wiki/Winged_edge

Since JavaScript does not support passing parameters by reference, you'll need to make the variable an object instead. You may assume that the input mesh is closed (i.e., no boundary) and there are no non-manifold vertices or edges in the mesh. Both input from and output to **OBJ** files should be implemented in this part.

Note that in the **OBJ** format, a mesh is given by a *vertex list* followed by a *face list*. Each line in the vertex list starts with the character 'v', followed by the *x*, *y*, and *z* vertex coordinates. Each line in the face list starts with the character 'f', and followed by three integers indexing into the vertex list. The vertex indexes start with 1 and are given in *counterclockwise order*, viewed from the tip of the triangle's outward pointing normal. Any comments start with the character '#'. The very first line of the **OBJ** file is of the form "# *n m*", where *n* is the number of vertices and *m* the number of faces in the mesh. Some sample useful **OBJ** files can be found in the assignment directory:

http://www.cs.sfu.ca/~haoz/teaching/cmpt464/assign/a1/

3. *[4 marks] Mesh display*

   Write code to display the mesh (in various modes, as specified in part 1) in the display box of your GUI. Note that you want to ensure proper width and stable visibility of the mesh edges. You may choose lighting and material parameters on your own. As far as efficiency is concerned, your program should be able to load and display meshes with up to 50,000 faces reasonably interactively. Obviously, you cannot rely on a quadratic-time OBJ parser. Try your program on larger models, e.g., the horse model with about 20,000 vertices.

4. Bonus problem [6 extra marks]: Butterfly and Loop subdivision

   You are challenged to implement the Butterfly and Loop (the original, not Warren's, as seen in Slide 16 of Lecture slides on Subdivision Zoo) subdivision schemes. If you do so, you should update your GUI to provide additional controls (e.g., another combo box to choose between Butterfly and Loop, a slider to choose number of subdivision levels, and a button to start the subdivision processing) in the GUI you implemented so that the results from your subdivision program can be properly generated and displayed.

   A simplifying assumption you can make is that the input mesh is a closed manifold mesh. Some small test meshes and results from subdivisions can be found in

   http://www.cs.sfu.ca/~haoz/teaching/cmpt464/assign/a1/subdivision

What to submit electronically:

All the **html**, **JavaScript** and **CSS** source code in a directory called **obj_view**. You should submit a **single ZIP file** (**a1q1.zip**), which when opened, will contain this directory with the appropriate files in it. The main file should be called **index.html**, opening it in a modern browser from the system directory should start the GUI so your implementation can be tested. Also submit a **README** file that contains at least the following information:

- Which feature(s) listed above were *not* implemented.
- Any special instructions or information you wish to provide for the grader.

If you are submitting code for the bonus question, similarly, please submit all the source code in a directory called subd. Make the zip file **a1bonus.zip**. The executable program should also be called **index.html**. Also submit a **README** as above.

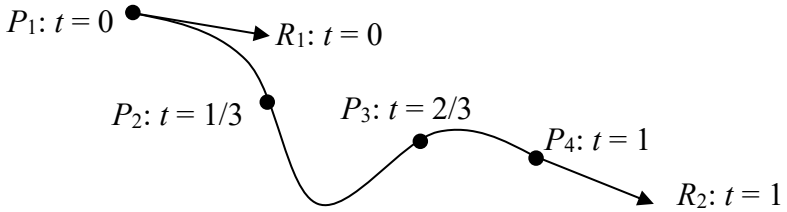
Problem 2 (2 marks): Sampling for surface reconstruction

First review slides 63-65 of the lecture on 3D shape representations. During the lecture, I mentioned that over a flat region of a surface, we do not need a lot samples to ensure an accurate surface construction, is this really true? Explain your answer. **Hint:** use lfs.

Problem 3 (2 marks): Parametric curve design

Determine the change of basis matrix for a quintic (that is, degree-5) parametric curve which is defined by four points and two tangent vectors, as shown below. Use the transpose

of $[P_1\ R_1\ P_2\ P_3\ P_4\ R_2]$ as your vector of control points or observable quantities. Expressing your solution as the *inverse of a computed matrix* is sufficient.



Problem 4 (3 marks): Continuity of cubic B-splines

Think about how the change-of-basis matrix for cubic B-splines can be derived. Note that you *need not* submit a solution for this. You need to complete the following however. Recall that given the vector of four control points $P = [P_0\ P_1\ P_2\ P_3]^T$ and the monomial basis $T = [1\ t\ t^2\ t^3]$, the cubic B-spline curve piece defined by P is given by

$$p(t) = TM_{\text{B-spline}}P, \text{ where } M_{\text{B-spline}}$$

is the cubic B-spline change-of-basis matrix,

$$M_{B-spline} = \frac{1}{6}\begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

Prove that piecewise cubic B-spline curves are $\mathbf{C}^2$.

Problem 5 (3 marks): Affine maps

Let $\boldsymbol{x}$ be a 3-by-1 column vector representing a 3D point. An *affine map* of $\boldsymbol{x}$ is defined by the following transformation: $F(\boldsymbol{x}) = P\boldsymbol{x} + \boldsymbol{q}$, where P is a 3-by-3 matrix and $\boldsymbol{g}$ is a 3-by-1 column vector. The new point $F(\boldsymbol{x})$ is said to be the *affine image* of $\boldsymbol{x}$. Affine maps include translation, rotation, shear, scaling, parallel projections, etc.

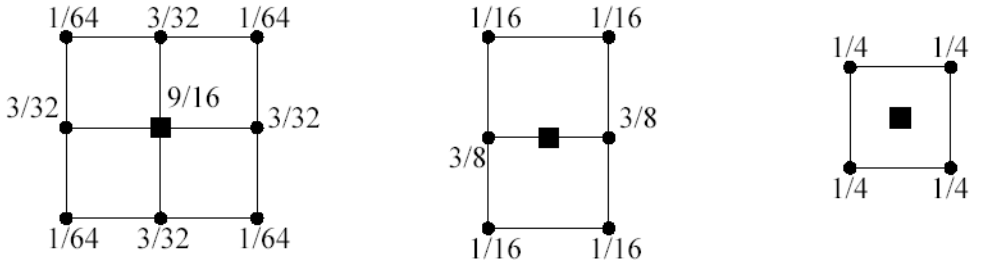
Consider a space curve C defined by a set of control points V. One way to find an affine image $F(C)$ of C is to transform all the points of C by F directly. Another way is to use F to transform the control points only, yielding the new set of control points $F(V)$. The affine image $F(C)$ of C is then constructed from the new control points, in the same way C was constructed from V. If the same affine image curve is obtained by both methods, we say the corresponding curve representation is *affine invariant*. Affine invariance is desirable in computer graphics and computer-aided geometric design applications.

Prove that cubic Bezier curves are affine invariant. **Hint:** You should first formulate the problem mathematically. If you do this properly, the proof is quite simple.

Problem 6 (2 marks): Midpoint subdivision

Refine an arbitrary closed manifold triangle mesh, whose connectivity is characterized by the graph G, by subdividing each triangle uniformly into four sub-triangles. Call the resulting graph G_1. Then compute the centroids of all the sub-triangles. Connect these centroids in the fashion of a dual graph G'_1 of G_1. This results in a polygon mesh composed mostly of hexagons. Finally, compute the centroids of all these polygons and connect them in the fashion of a dual graph G''_1 of G'_1. These constitute one step of a *midpoint subdivision scheme* for arbitrary triangle meshes.

Compute the *masks* (showing topological and geometric rules of subdivision, as given in the figure below for Catmull-Clark subdivision over regular regions) for both regular and (general) extraordinary cases of this scheme.



Vertex rule Edge rule Face rule