Assignment 3 - Geometry

This assignment makes you practice geometric algorithms in computer graphics. For your in-depth understanding of this area, it also covers a few related concepts that might new to you. You should use any resources (e.g., books, search engines, calculators, and etc.) that may help you accomplish it.

The assignment provides a few code skeletons for corresponding tasks, please check `https://github.com/mimuc/cg1-ss20`.

Task 1: Geometric Representations

There is no "best" representation. Some geometric representation work better depending on the task. For instance, it is not a good decision to use polygon mesh to represent fluids. This task helps you walk through different geometric representations.

a) Given a sphere (A) and a cuboid (B), the Constructive Solid Geometry (CSG) operations result in geometries using AND ($\cap$), OR ($\cup$), NOT ($-$), and XOR ($\oplus$) that are shown in Figure 1. Write the corresponding CSG formula for each subfigure.

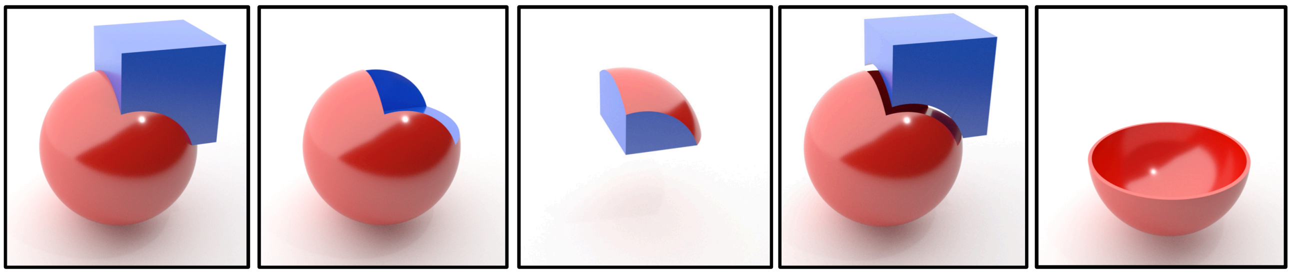
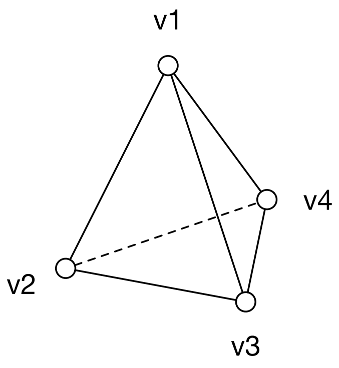


Figure 1: CSG operations were performed in each of the five figures.



Vertex List

v1	
v2	
v3	
v4	

Face List

Figure 2: Using a Face-Vertex table to represent a tetrahedron.

b) Figure 2 shows a tetrahedron with 4 given vertices. Fill the vertex list and face list table (coordinates can be omitted) using Face-Vertex mesh representation to express the given tetrahedron.

A hilly terrain is usually represented as an unflattened surface, as illustrated in Figure 3, as demonstrated in this demo[1]. It is common to use noise (e.g. Perlin noise[2]) for generating random content, a terrain is a classic example.

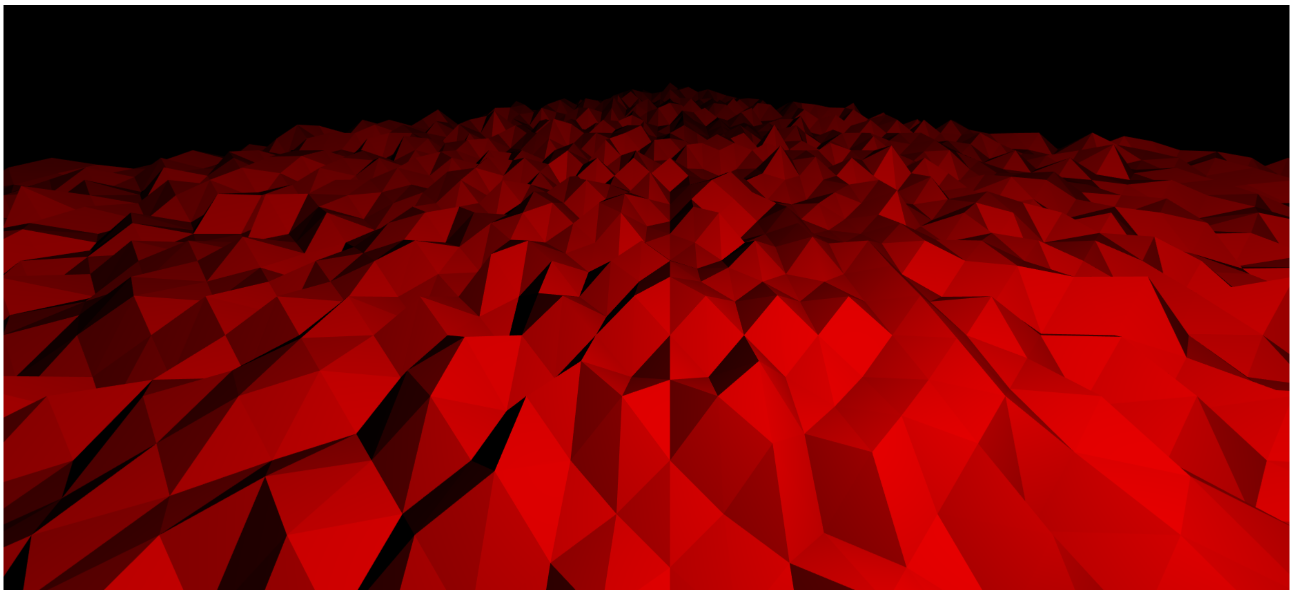


Figure 3: An example of a hilly terrain.

c) To implement the terrain, which representation is more suitable than others?

d) Which geometry do you need for the implementation of the terrain, and how do you modify this object for creating the terrain?

e) In the code skeleton, you can find four files: `main.js`, `renderer.js`, `perlin.js`, and `terrain.js`. Use the given Perlin noise generator to implement the `// TODO:` in `terrain.js`.

f) Why are triangles mostly used as basic elements comparing to complex polygons in 3D rendering? Name three reasons.

g) Why are quad-meshes commonly used in 3D modeling? Name three reasons.

Answer text questions in the `README.md` of the code skeleton, then include your answers and implementation in a folder called “task01”. ***Exclude*** the installed dependencies (folder `node_modules`) in your submission.

[1] http://www.medien.ifi.lmu.de/lehre/ss20/cg1/demo/3-geometry/terrain/
[2] http://en.wikipedia.org/wiki/Perlin_noise

Task 2: Bézier Curves Interpolation

In the lecture, you learned *Bézier Curves* and the basic idea of the *de Casteljau algorithm* regarding how to draw a Bézier curve recursively. Let's now go from theory to practice, and implement it eventually.

The control points $P_0(1,4)$, $P_1(3,1)$, $P_2(6,2)$, $P_3(8,6)$ of a cubic Bézier curve are shown in Figure 4.

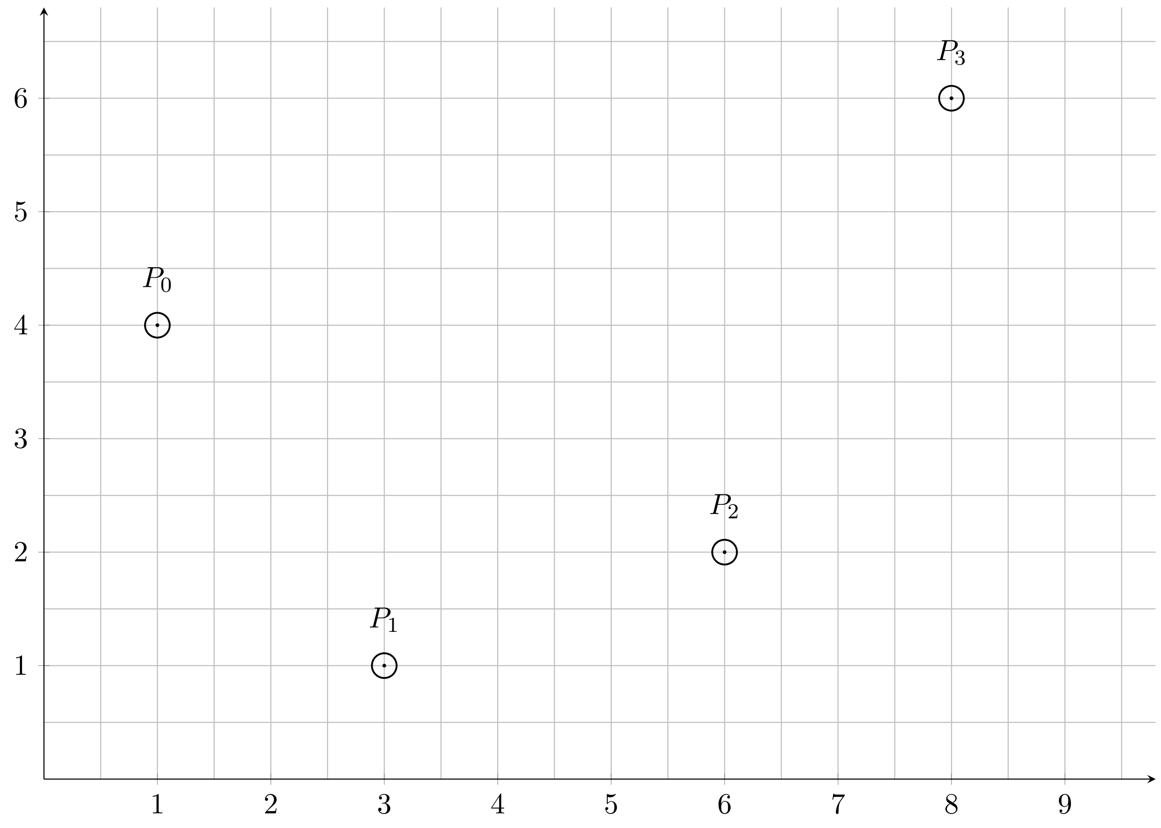


Figure 4: Control points of a cubic Bézier curve

a) Draw the point on the Bézier curve determined by the given control points where parameter $t = 0.5$.

To implement the de Casteljau algorithm, we need to calculate the formula for each point on a Bézier curve.

b) Given two points $\mathbf{b}_0(x_0, y_0), \mathbf{b_1}(x_1, y_1)$. Show $\mathbf{b}_0^1(x, y)$ that

$$\frac{|\mathbf{b}_0\mathbf{b}_0^1|}{|\mathbf{b}_0^1\mathbf{b}_1|} = \frac{t}{1-t}, t \in [0,1]$$

where $|\cdot|$ represents the distance between two points.

c) In the provided code skeleton, you can find three JS files: `main.js`, `renderer.js`, and `bazier.js`. Combining with the results you found in b), look for `// TODO:` in `bazier.js`, and implement the de Casteljau algorithm. If your implementation is correct, your scene should look similar to Figure 5. There is an interactive demo[3] to help you understand the scene, you can move the camera using the right button of your mouse.

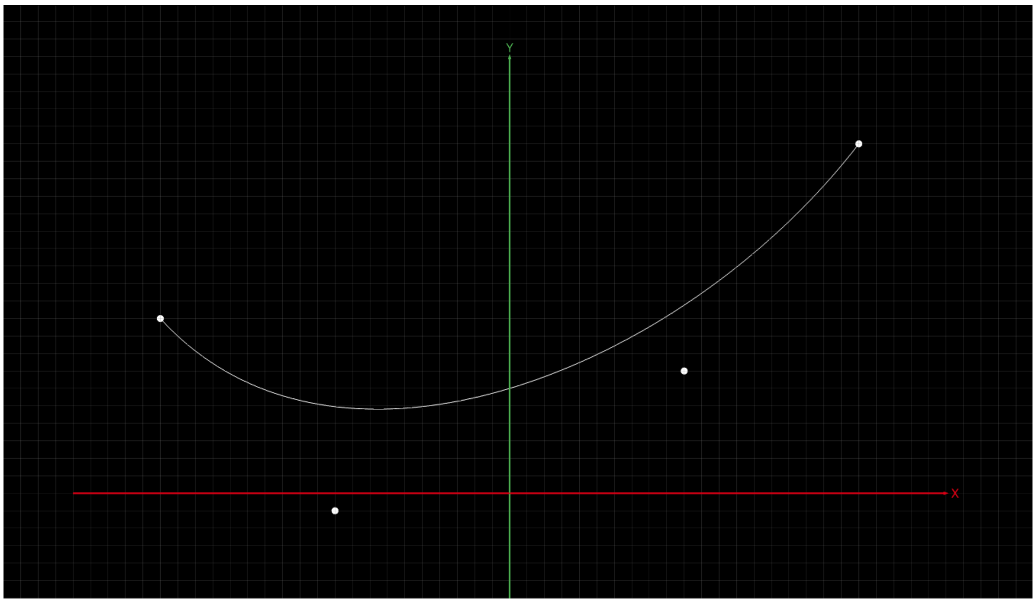


Figure 5: Final scene of the de Casteljau algorithm implementation.

d) Name three properties of Bézier curves.

e) What constraints two connected piecewise Bézier curves to be "seamless"?

f) Name a reason why do we need piecewise Bézier curves other than a single curve with many control points?

g) Why do we have 16 control points for a bicubic Bézier patch?

Answer text questions in the `README.md` of the given code skeleton, then include your answers and implementation in a folder called "task02". ***Exclude*** the installed dependencies (folder `node_modules`) in your submission.

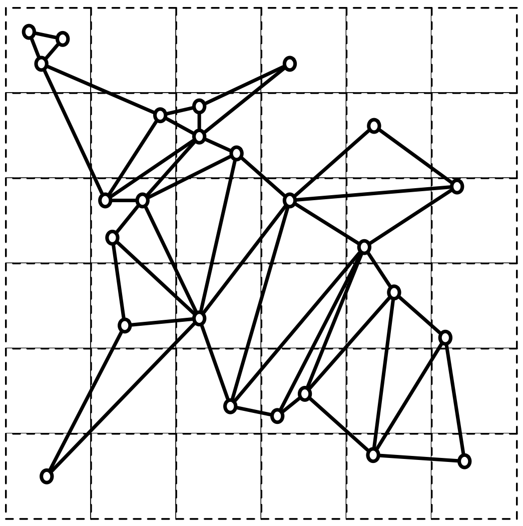
Task 3: Mesh Sampling

In the lecture, you learned the basic idea of the algorithm *vertex clustering* is to perform *mesh simplification* by *downsampling*. To practice the algorithm, a mesh is given in the Figure 6a.

a) Apply the vertex clustering algorithm to draw a simplified mesh in Figure 6b.

b) What is the reduction ratio in terms of the number of triangles?

[3] `http://www.medien.ifi.lmu.de/lehre/ss20/cg1/demo/3-geometry/bezier/`


(a) A given mesh for further simplification operation.

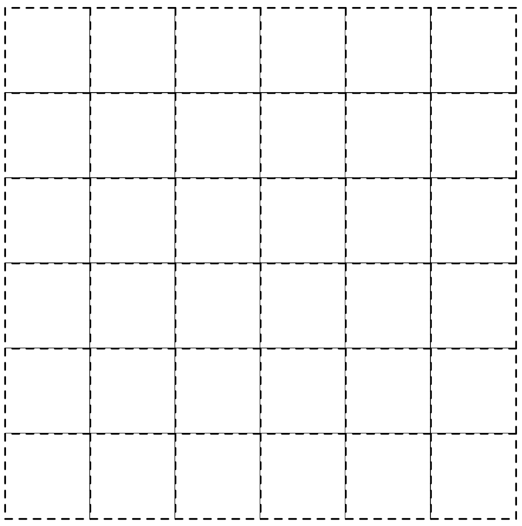

(b) Empty template for you to draw.

Figure 6: Apply vertex clustering algorithm.

c) What drawback of the algorithm shows in your result?

The goal of mesh simplification is to reduce the number of polygons while preserving the overall shape. In contrast, one can increase the number of polygons using *subdivision* for *upsampling*. Mesh subdivision subdivides each polygon into smaller faces that better approximate a smooth surface.

An interesting question: What could happen if we repeat these two operations as shown in Figure 7?

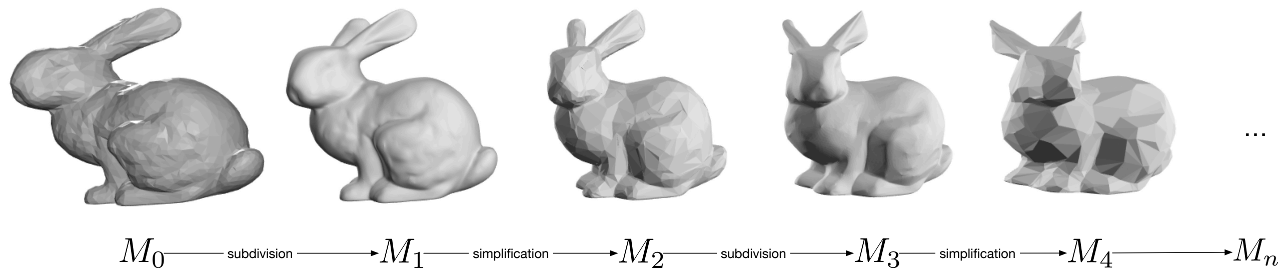


Figure 7: Repeating mesh subdivision and simplification.

There is an interactive demo[4] for you to explore. Implement this process in the provided code skeleton using the existing `three.js` implimentation: `SubdivisionModifier`[5] and `SimplifyModifier`[6].

[4] http://www.medien.ifi.lmu.de/lehre/ss20/cg1/demo/3-geometry/bunny/
[5] https://threejs.org/examples/?q=modifier#webgl_modifier_simplifier
[6] https://threejs.org/examples/?q=modifier#webgl_modifier_subdivision

d) What algorithms are used in `three.js` regarding mesh simplification and subdivision?

e) Assume the subdivision number is 2, and the reduction ratio of number of vertices is 95%. In the code skeleton, you can find `main.js`, `renderer.js`, and `bunny.js`. Implement the `// TODO:` in `bunny.js`. Then present a picture for M_{10}.

f) What did you observed from your result and what can you conclude from it?

Answer text questions in the `README.md` of the provided code skeleton, then include your answers and implementation in a folder called "task03". ***Exclude*** the installed dependencies (folder `node_modules`) in your submission.

Submission

- Participation in the exercises and submission of the weekly exercise sheets is voluntary and not a prerequisite for participation in the exam. However, participation in an exercise is a good preparation for the exam (the content is the content of the lecture and the exercise).
- For non-coding tasks, write your answers in a Markdown file. Markdown is a simple mark-up language that can be learned within a minute. A recommended the Markdown GUI parser is typora (`https://typora.io/`), it supports parsing embedded formula in a Markdown file. You can find the syntax reference in its Help menu.
- **Please submit your solution as a ZIP file** via Uni2Work (`https://uni2work.ifi.lmu.de/`) before the deadline. We do not accept group submissions.
- Your solution will be corrected before the discussion. Comment your code properly, organize the code well, and make sure your submission is clear because this helps us to provide the best possible feedback.
- If we discover cheating behavior or any kind of fraud in solving the assignments, you will be withdrawn for the entire course! If that happens, you can only rejoin the course next year.
- If you have any questions, please discuss them with your fellow students first. If the problem cannot be resolved, please contact your tutorial tutor or discuss it in our Slack channel.