Prof. Dr. Leif Kobbelt
Patrick Schmidt, Joe Jakobi

Visual Computing Institute | RWTH AACHEN UNIVERSITY

Basic Techniques in Computer Graphics

Assignment 6

Date Published: November 29th 2022, Date Due: December 6th 2022

- All assignments (programming and theory) have to be completed in teams of 3–4 students. Teams with fewer than 3 or more than 4 students will receive no points.
- Hand in **one solution per team per assignment**.
- Every team must work independently. Teams with identical solutions will receive no points.
- Solutions are due on December 6th 2022 via Moodle. Late submissions will receive zero points. No exceptions!
- Instructions for **programming assignments**:
  - Make sure you are part of a Moodle group with 3-4 members. See "Group Management" in the Moodle course room.
  - Download the solution template (a zip archive) through the Moodle course room.
  - Unzip the archive and populate the `assignmentXX/MEMBERS.txt` file. The names and student ids listed in this file **must match** your moodle group **exactly**.
  - Complete the solution.
  - Prepare a new zip archive containing your solution. It must contain exactly the files that you changed. **Only change the files you are explicitly asked to change in the task description.** The directory layout must be the same as in the archive you downloaded. (At the very least it must contain the `assignmentXX/MEMBERS.txt`.)
  - One team member uploads the zip archive through Moodle before the deadline, using the group submission feature.
  - Your solution must compile and run correctly **on our lab computers** by only inserting your **assignment.cc** and **shader files** into the Project. If it does not compile on our machines, you will receive no points. If in doubt you can test compilation in the virtual machine provided on our website.
- Instructions for **text assignments**:
  - Prepare your solution as a single pdf file per group. Submissions on paper will not be accepted.
  - If you write your solution by hand, write neatly! Anything we cannot decipher will receive zero points. No exceptions!
  - Add the names and student ID numbers of all team members to every pdf.
  - Unless explicitly asked otherwise, always justify your answer.
  - Be concise!
  - Submit your solution via Moodle, together with your coding submission.

Prof. Dr. Leif Kobbelt
Patrick Schmidt, Joe Jakobi

Visual Computing Institute | RWTH AACHEN UNIVERSITY

Exercise 1 Barycentric Coordinates [7 Points]

In the lecture, we learned how to compute the barycentric coordinates of a ray-triangle intersection point in a 3D setting. In this task, we compute the barycentric coordinates in 2D.

(a) Linear System [2 Points]

Given are a triangle defined by $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{R}^2$ and a point $\mathbf{p} \in \mathbb{R}^2$. Derive the 2-by-2 linear system (in the form $\mathbf{A}\mathbf{x} = \mathbf{r}$) that needs to be solved in order to compute the barycentric coordinates α and β of $\mathbf{p}$ in the triangle. Briefly explain your derivation.

(b) Closed-Form Solution [2 Points]

There are multiple ways to solve the equation. In this task, use Cramer's rule[1] to derive closed-form solutions for α and β, i.e. specify two formulas that compute α and β independently of each other.

(c) Example [3 Points]

Consider $\mathbf{a} = (2,2)^\mathsf{T}, \mathbf{b} = (10,2)^\mathsf{T}, \mathbf{c} = (2,4)^\mathsf{T}, \mathbf{p} = (3,3)^\mathsf{T}$. Compute the barycentric coordinates α, β, γ of $\mathbf{p}$ in the triangle $\mathbf{a}, \mathbf{b}, \mathbf{c}$. Write our your intermediate computations.

Exercise 2 Ray-Quadric Intersection [11 Points]

In the lecture you learned about a general way to find the intersection point(s) of a ray with a quadric surface. Consider the surface in $\mathbb{R}^3$ defined by the quadric

$$\mathbf{Q} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

(a) Surface Description [2 Points]

Describe the surface defined by $\mathbf{Q}$ in a few words (e.g. the type of object, its dimensions, its position and alignment in space).

(b) Ray Intersection [6 Points]

We now intersect the above surface with the ray $\mathbf{c}+\lambda\mathbf{r}$, where $\mathbf{c}$ is given as $(0,0,\sqrt{2})^\mathsf{T}$ and $\mathbf{r} = (r_x, r_y, r_z)^\mathsf{T}$ is a general direction. Derive a closed form solution for the parameter λ at the intersection point(s), i.e. a formula $\lambda(r_x, r_y, r_z) \in \mathbb{R}$. **Explain your derivation** step by step!

(c) Example [3 Points]

Now also consider $\mathbf{r}$ to be given as $(0,1,-1)^\mathsf{T}$ and compute the intersection points $\mathbf{p}_1, \mathbf{p}_2 \in \mathbb{R}^3$. How many distinct intersection points exist? What is the geometric interpretation of this configuration?
As the $\pm$ term cancels, there is only one distinct solution:

$$\mathbf{p}1 = \mathbf{p}2 = \begin{pmatrix} 0 \\ 0 \\ \sqrt{2} \end{pmatrix} + \frac{\sqrt{2}}{2} \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{pmatrix}$$

The ray tangentially touches the cylinder at a single point.

[1] https://en.wikipedia.org/wiki/Cramer%27s_rule

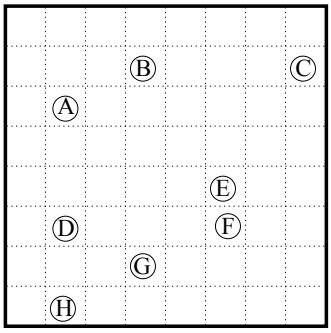
Prof. Dr. Leif Kobbelt
Patrick Schmidt, Joe Jakobi

Exercise 3 Spatial Data Structures [7 Points]

To speed up algorithms a scene is often partitioned using spatial data structures.

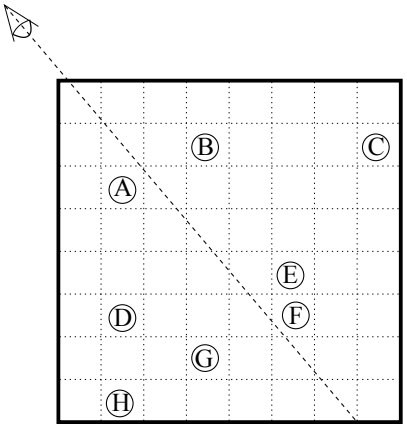
(a) Quad-Tree [3 Point]

Construct a quad-tree for the scene shown below. Stop subdivision as soon as every node contains at most one object. Draw the splitting lines into the illustration of the scene and sketch the resulting tree structure.



(b) kD-Tree [3 Point]

Construct a kD-tree for the scene shown below. Stop subdivision as soon as every node contains at most one object. Start with a split parallel to the horizontal axis. Draw the splitting lines into the illustration of the scene and sketch the resulting tree structure.



(c) Painter's Algorithm [1 Point]

Use the kD-tree constructed in the previous task to determine the order in which the objects are drawn during Painter's Algorithm, given the camera position shown in the illustration.

Exercise 4 2D Polygon Triangulation: Ear Clipping [15 Points]

In this task you are going to implement the ear clipping algorithm presented in the lecture. The algorithm takes a simple planar polygon as input and returns a set of triangles, covering the same area.

The interface is realized by a function `void triangulate(const std::vector<glm::vec2>& vertices, std::vector<int>& triangles)`. Here, `vertices` contains a list of 2D points in which each pair of consecutive points defines an edge.

The result is returned via the vector `triangles`, which is a list of integers. Each integer is an index into the list of vertices and three consecutive vertex indices define a triangle. I.e. the length of `triangles` is three times the number of resulting triangles. Initially `triangles` is empty.

Remember that the ear clipping algorithm works by successively checking if a vertex can be removed from the polygon by connecting its two neighboring vertices with an edge (clipping away an ear). Vertices can only be clipped if they are a convex corner of the remaining polygon and no other vertex lies inside the new triangle. Fortunately, such a vertex always exists and the algorithm terminates when only two vertices are left.

You can assume that all input polygons are defined in a counter-clockwise manner. I.e. you do not have to find out which sign corresponds to concave or convex corners. Also define the resulting triangles in counter-clockwise order!

By pressing the keys `A`, `B`, `C`, `D`, `E`, you can switch between different test polygons. Your algorithm is executed each time you switch to a new example. Figure 1 shows possible triangulations for some test cases. Note that different valid solutions are possible.

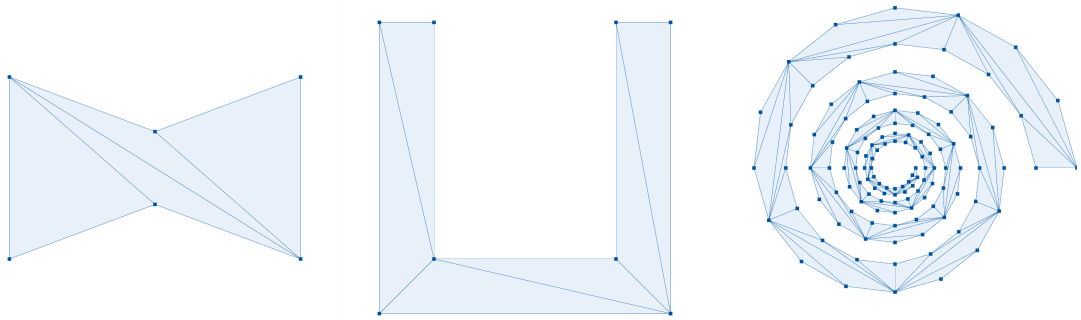


Figure 1: Possible triangulations for the test cases in scenes (a), (c) and (e).

(a) Convex Corners [2 Points]

Implement the function `bool convex(glm::vec2 prev, glm::vec2 curr, glm::vec2 next)`, that checks if a given corner is convex. The corner is defined by the vertex `curr` and its previous and next neighbors in counter-clockwise order. Also return false if all three vertices are collinear.

(b) In-Triangle Test [2 Points]

Implement the function `bool inTriangle(glm::vec2 p, glm::vec2 a, glm::vec2 b, glm::vec2 c)`, that checks if the vertex `p` is inside the triangle a, b, c. Also return true if `p` lies on the boundary of the triangle.

(c) Empty Triangle [2 Points]

Further, implement the function `bool triangleEmpty(int i_a, int i_b, int i_c, std::vector<glm::vec2>& vertices)`, checking if any vertex of the given vector is inside the triangle a, b, c. Here, `i_a`, `i_b`, `i_c` are indices of the `vertices` vector. Use the function `inTriangle(...)` you implemented in part (b). Make sure to exclude the cases of checking whether a, b or c themselves lie inside the triangle.

(d) Ear Clipping Algorithm [9 Points]

With those helper functions in place, implement the ear clipping algorithm in the function `triangulate(...)` and verify its behavior in all 5 test cases.

Hints:

- We already created the vector `clipped`, which you can use to mark vertices as deleted. Your algorithm should simply skip those vertices instead of deleting them from the input list.
- You can use a while loop to iterate around the remaining polygon and stop when there are only two vertices left. Keep indices of the current, previous and next (non-deleted) vertex and advance them at the end of each iteration. Watch out for infinite loops.
- For this exercise you do *not* have to optimize the run time performance of your algorithm by choosing a particular order for testing vertices.
- Watch the terminal output. We print a message in a few typical error cases.
- A simple way to debug your algorithm is to print test messages or the value of some variables to the terminal using `std::cout`. We already added an example to the existing code.