# Assignment 5 - Rasterization

This assignment contains topics related to rasterization, as the most important concept in classic computer graphics.

There are code skeletons for programming tasks, please check our GitHub repository [1].

## ~~Task~~ 0: Midterm Evaluation Survey

We value your learning experience in having the course entirely online, and expect to understand your status. With your feedback, we could make better plans for the rest of the summer semester. Please complete this anonymous survey: https://forms.gle/XqWC5cctM56GBvZV9

Thank you very much for your feedback :)

## Task 1: Rasterization

a) What types of culling were discussed in the lecture? Explain each of them in one sentence.

b) What is the purpose of clipping? What issue can arise with clipping?

c) Name a clipping algorithm for line clipping, then explain how it works.

d) What issue can arise with a depth buffer?

e) For what reason is the issue occuring in d)? How to solve it?

f) What is frame buffer? Why do you need it?

In the lecture, you learned the **Bresenham algorithm** for drawing lines and the **Scan Line algorithm** for drawing polygons. Let's implement them for lines and triangles.

In the given code skeleton, you can find three JS files: `main.js`, `renderer.js`, and `app.js`. We have created a grid plane intended to mock a pixel-based screen in the provided code skeleton. You should use the implemented method `drawPoint(x, y, color)` in `app.js` for shading a pixel in the grid.

g) Look for `// TODO:` in `app.js`, implement the Bresenham and Scanline algorithm for all given geometry primitives in `app.js`. A correct implementation will look similar to Figure 1. An interactive demo can be found online[2].

h) What issue can occur with the Bresenham algorithm?

i) Name an algorithm that was proposed to address the issue in h) then explain how it works.

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

[1] https://github.com/mimuc/cg1-ss20
[2] https://www.medien.ifi.lmu.de/lehre/ss20/cg1/demo/5-raster/bresenham/

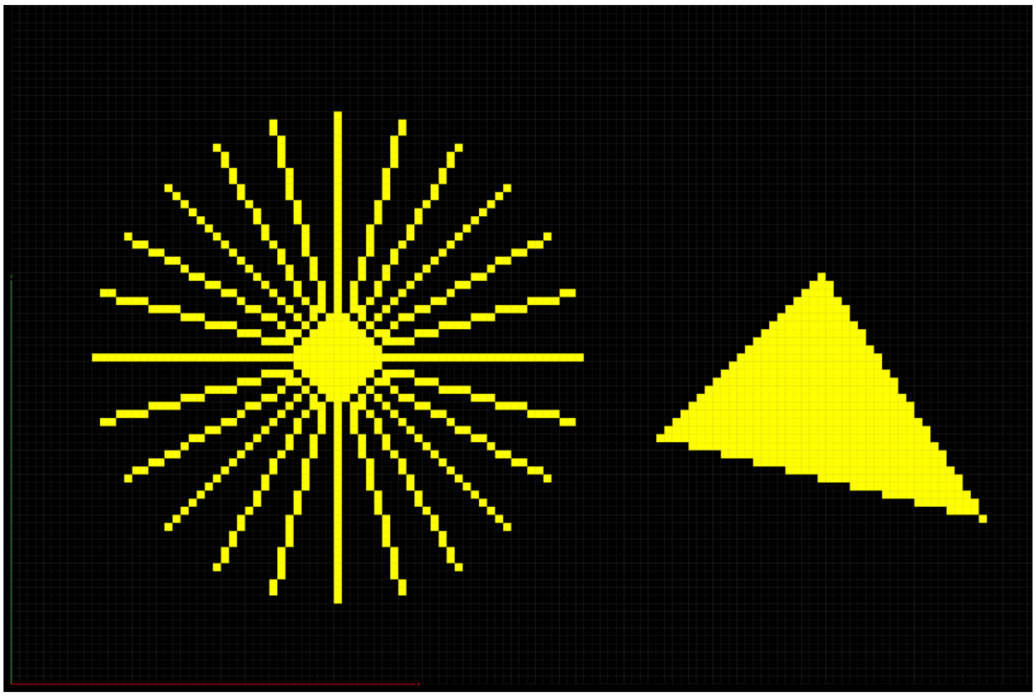

Figure 1: The final scene of drawing triangle and lines.

Congratulations! So far with all previous assignments, you have experienced all important steps in the transformation and rasterization pipeline from 3D geometry to a pixel-based 2D image. Now you should be able to write a rasterization pipeline on your own. What else do you need to generate a photorealistic image or animations? Starting from this task, you are going to practice the other critical aspect of computer graphics: rendering.

## Task 2: Getting started with GLSL

In the lecture, you learned the OpenGL Shading Language (GLSL) as an example of shading languages. In this task, we are going to walk you through different concepts to work with GLSL, read an additional tutorial document might be helpful[3].

a) What do vertex and fragment shaders do? What is the compute shader used for?

b) What properties do variables of the type `uniform` have? Where are they set, where are they available? Does the value of a `uniform` variable differ for different vertices? Give an example of a typical `uniform` variable.

c) What properties distinguish `attribute` variables? Where can these variables be accessed? Where are they set and where can their contents be changed?

[3] `https://learnopengl.com/Getting-started/Shaders`

d) What is the special feature of `out` variables in the vertex shader? Describe what they are used for and what happens with their values.

e) Which predefined variable must be written in the vertex shader?

f) What effect does the `out` variable have in the fragment shader?

In `three.js`, you can create custom shader by using `ShaderMaterial`[4] or `RawShaderMaterial`[5] that include your customized vertex shader and fragment shader. The difference is that `Shadermaterial` provides several built-in attributes and uniforms by `WebGLProgram`[6].

In the code skeleton, you can find two JS files: `main.js` and `renderer.js`; and two GLSL files: `vert.glsl` and `frag.glsl`.

g) Look for `// TODO:` in `main.js`, `vert.glsl`, and `frag.glsl`. First, create a `Geometry`[7] that contains three vertices and a `Face3`[8]. Specify the color of each vertex in `main.js`. Afterwards, create a `ShaderMaterial` with a loaded vertex shader and fragment shader, then build a `Mesh` with the created geometry. In the vertex shader, scale the geometry on the x-axis by 1.5, the y-axis by 0.5, and the z-axis by 2; receive the color from three.js then pass it to the fragment shader. In the fragment shader, receive the color from the vertex shader and pass it as the output color. A demo can be found online[9].

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

[4] `https://threejs.org/docs/#api/materials/ShaderMaterial`
[5] `https://threejs.org/docs/#api/en/materials/ShaderMaterial`
[6] `https://threejs.org/docs/#api/en/renderers/webgl/WebGLProgram`
[7] `https://threejs.org/docs/#api/core/Geometry`
[8] `https://threejs.org/docs/#api/core/Face3`
[9] `https://www.medien.ifi.lmu.de/lehre/ss20/cg1/demo/5-raster/shader/`

## 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.