RAS

RT

neural rendering?

nerf

Gaussian splatting

遅い

難しいから?

RENDERING PIPELINE

Introduction

The process of generating an image given

絶対分

- a virtual camera
- objects
- light sources

It uses various techniques, e.g.

- rasterization (topic of this course)
- raytracing

One of the major research topics in computer graphics

- rendering
- animation three.js, ベーシック
- geometry processing

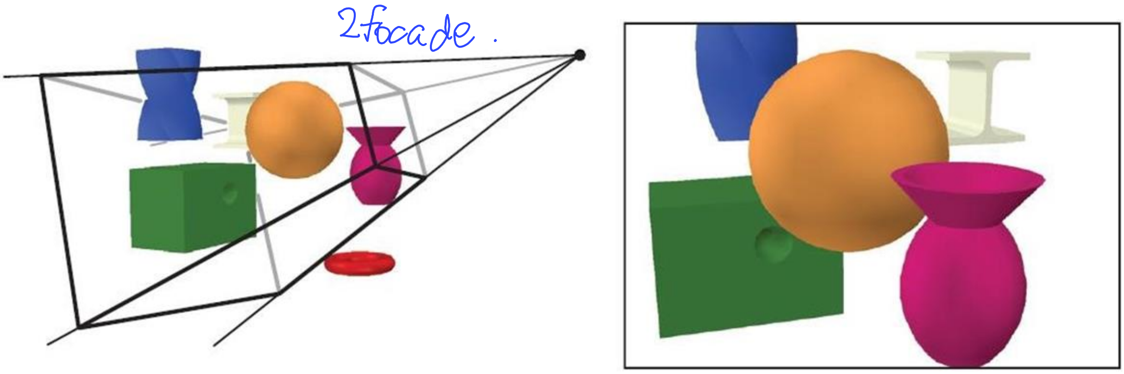
大抵, 動きは ほっぽの生成より かんたん (シミュレーションで物理演算など)

Rasterization

60 million/5

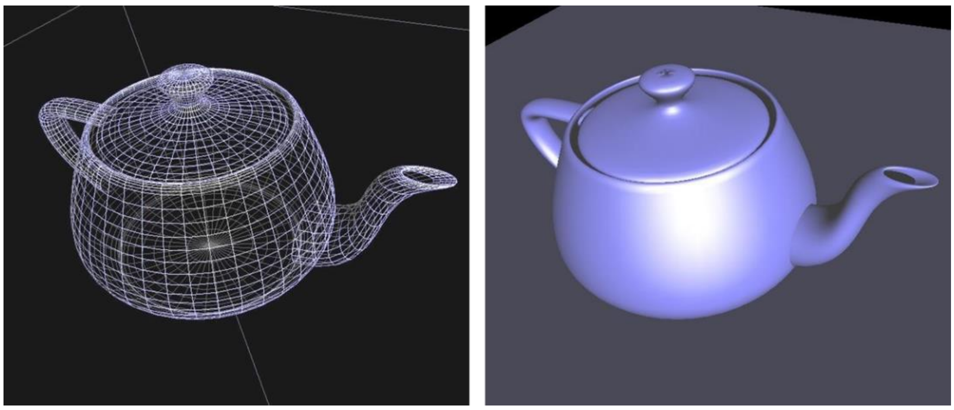
- Rendering algorithm for generating 2D images from 3D scenes.
- Transforming geometric primitives such as lines and polygons into raster image representations, i.e. pixels.

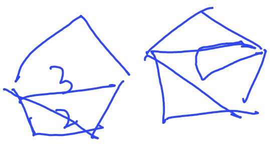
2 facade.



- 3D objects are approximately represented by vertices (points), lines, polygons
- These primitives are processed to obtain a 2D image.

殆どのものはポリゴンで表現される





ラスタリゼーション

Rendering pipeline

かざい処理のは: Pararel Processing → Multipul pararel operation - Multipul SIMD

Introduction

Processing stages comprise the rendering pipeline (graphics pipeline)

Supported by commodity graphics hardware

- GPU - graphics processing unit
- computes stages of the rasterization-based

All pararel

Rendering pipeline

シェーダー

- OpenGL (WebGL), DirectX, Vulkan, Metal (MAC) are software interfaces to graphics hardware
- this course focuses on concepts of the rendering pipeline
- this course assumes WebGL in implementation-specific details

Task

1/30

3D input

- A virtual camera
  - position, orientation, focal length
- Objects
  - points (vertex / vertices), lines, polygons
  - geometry and material properties (position, normal, color, texture coordinates)
- Light sources
  - direction, position, color, intensity
- Textures (images)

two vertices can write line

Lineは3つ必要だけど 6 点は6つ

2D output

Per-pixel color values in the framebuffer

pipelineに計画的に処理の内容で方法を情報を GPUに送る。

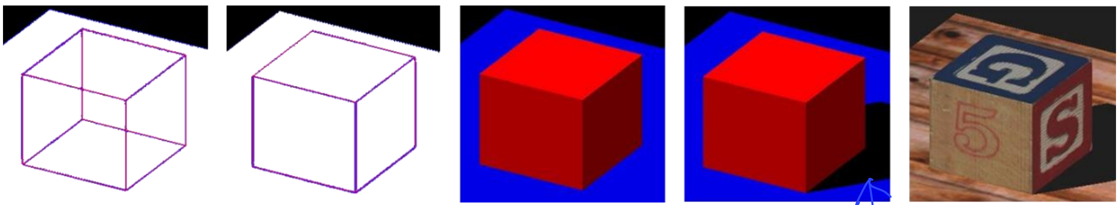
main component boost GPU

Geomety is already inside GPU

Some functionality

Sheder programing

- Resolving visibility
- Evaluating a lighting model
- Computing shadows (not core functionality)
- Applying textures



かげある。

Main Stages

Vertex processing / geometry stage / vertex shader

- processes all vertices independently in the same way
- performs transformations per vertex, computes lighting per vertex

Geometry shader

- generates, modifies, discards primitives

Primitive assembly and rasterization / rasterization stage

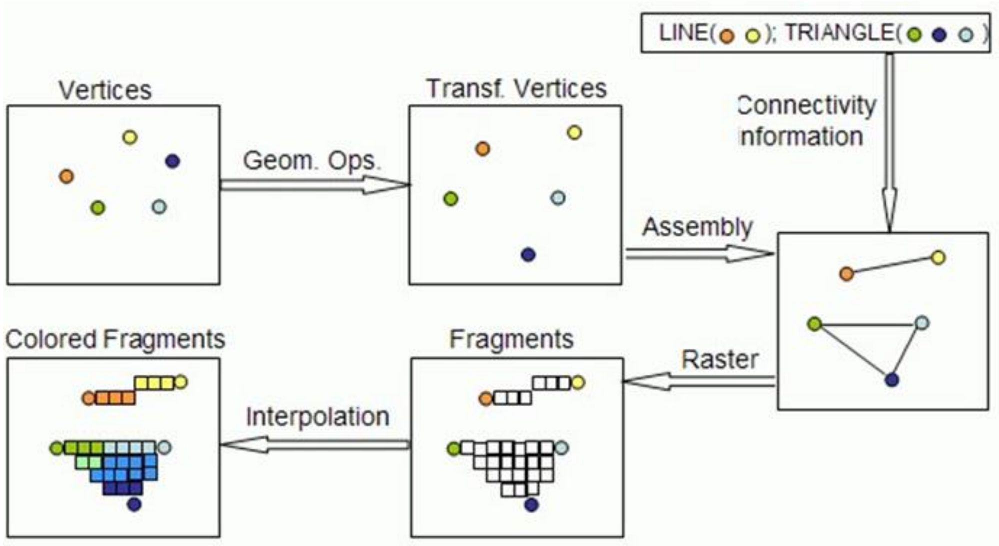
ラスタリゼーション

interporation

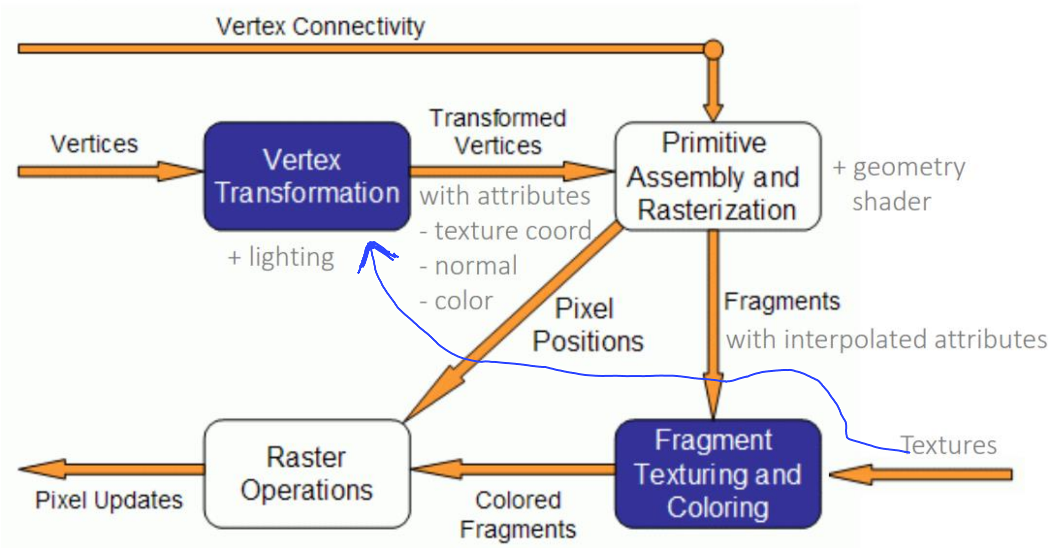
- assembles primitives such as points, lines, triangles
- converts primitives into a raster image
- generates fragments / pixel candidates
- fragment attributes are interpolated from vertices of a primitive

Fragment processing / fragment shader

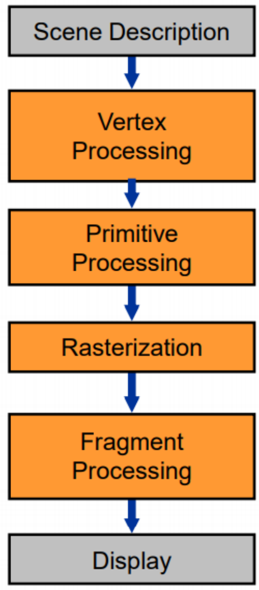
- processes all fragments independently in the same way
- fragments are processed, discarded or stored in the framebuffer



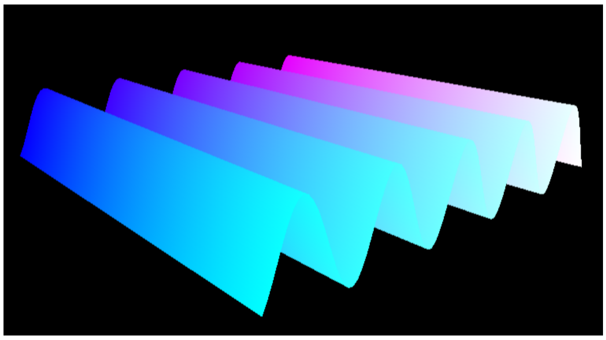
- Vertex position transform
- Lighting per vertex
- Primitive assembly, combine vertices to lines, polygons
- Rasterization, computes pixel positions affected by a primitive
- Fragment generation with interpolated attributes, e.g. color
- Fragment processing (not illustrated), fragment is discarded or used to update the pixel information in the framebuffer, more than one fragment can be processed per pixel position



Summary



hot plane



頂点

Vertex Processing

Geometry stage

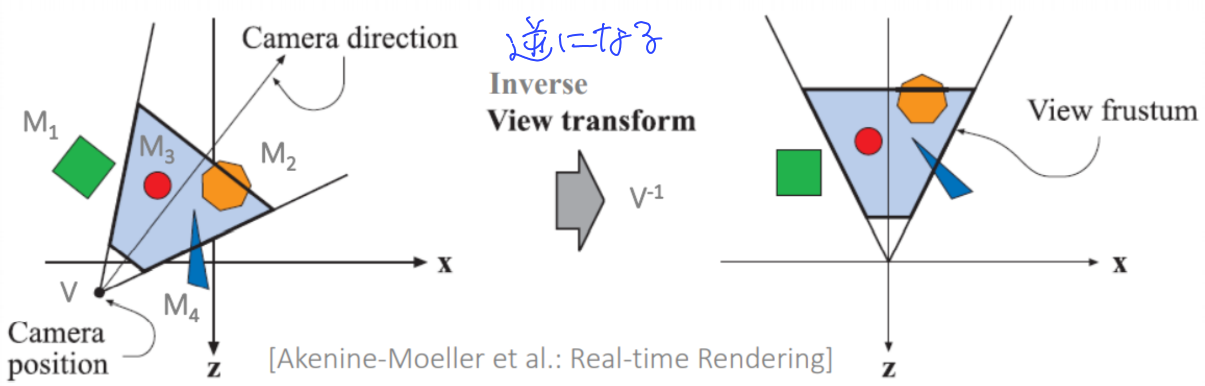
- Model transform
- View transform
- Lighting
- Projection transform
- Clipping
- Viewport transform

```
var geometry = new THREE.PlaneGeometry(1, 1, 100, 100);
```

10 thousand triangles.

Model Transform / View Transform

- Each object and the respective vertices are positioned, oriented, scaled in the scene with a model transform
- Camera is positioned and oriented, represented by the view transform
- I.e., the inverse view transform is the transform that places the camera at the origin of the coordinate system, facing in the negative z-direction
- Entire scene is transformed with the inverse view transform



[Akenine-Moeller et al.: Real-time Rendering]

canonical 正統的
manipulation 操作

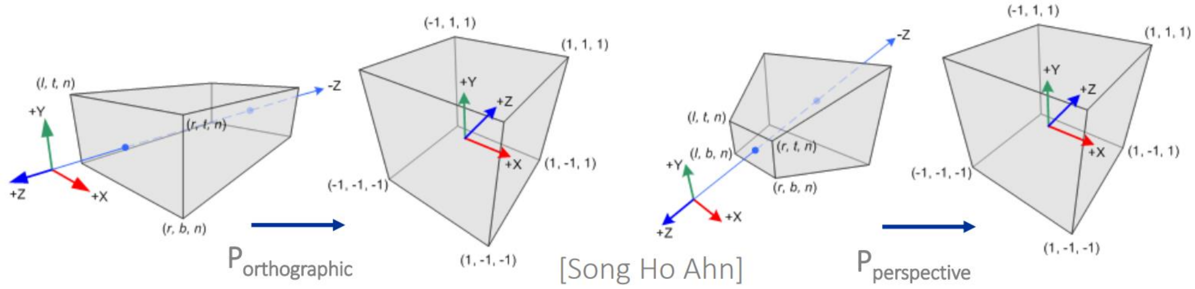
- M_1 , M_2 , M_3 , M_4 , V are matrices representing transformations
- M_1 , M_2 , M_3 , M_4 are model transforms to place the objects in the scene
- V places and orientates the camera in space
- V^{-1} transforms the camera to the origin looking along the negative z-axis ν model and view transforms are combined in the modelview transform ν the modelview transform $V^{-1}M_{1..4}$ is applied to the objects

Lighting

- Interaction of light sources and surfaces is represented with a lighting / illumination model
- Lighting computes color for each vertex
    - based on light source positions and properties
    - based on transformed position, transformed normal, and material properties of a vertex

Projection transform

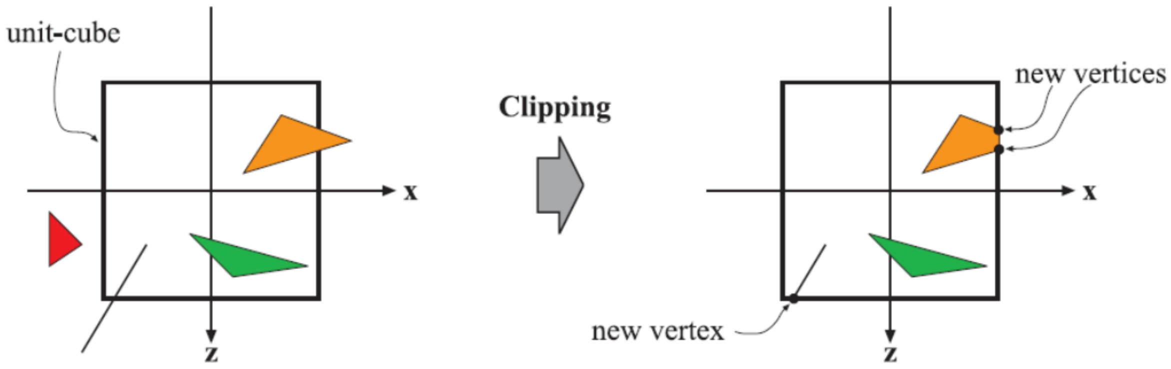
- Ptransforms the view volume to the canonical view volume
- The view volume depends on the camera properties
    - orthographic projection: cuboid
    - perspective projection: pyramidal frustum



- Canonical view volume is a cube from (-1,-1,-1) to (1,1,1)
- View volume is specified by near, far, left, right, bottom, top
- View volume (cuboid or frustum) is transformed into a cube (canonical view volume)
- Objects inside (and outside) the view volume are transformed accordingly
- Orthographic
    - combination of translation and scaling
    - all objects are translated and scaled in the same way
- Perspective
    - complex transformation
    - scaling factor depends on the distance of an object to the viewer
    - objects farther away from the camera appear smaller

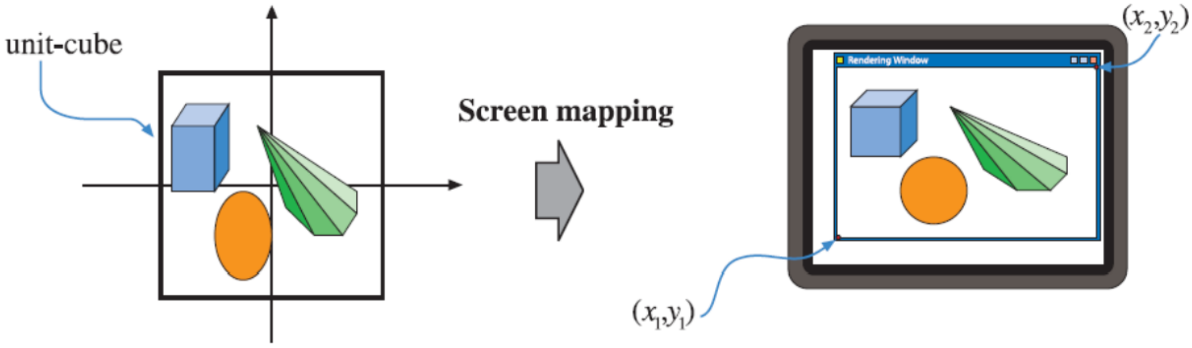
Clipping

- Primitives, that intersect the boundary of the view volume, are clipped
    - primitives, that are inside, are passed to the next processing stage
    - primitives, that are outside, are discarded
- Clipping deletes and generates vertices and primitives



Viewport Transform / Screen Mapping

- Projected primitive coordinates (X_p , Y_p , Z_p) are transformed to screen coordinates (X_s , Y_s)
- Screen coordinates together with depth value are window coordinates (X_s , Y_s , Z_w)



- (X_p , Y_p) are translated and scaled from the range of (-1, 1) to actual pixel positions (X_s , Y_s) on the display
- Z_p is generally translated and scaled from the range of (-1, 1) to (0,1) for Z_w
- Screen coordinates (X_s , Y_s) represent the pixel position of a fragment that is generated in a subsequent step
- Z_w , the depth value, is an attribute of this fragment used for further processing

Summary

object space

⇓ modelview transform

eye space / camera space

⇓ lighting, projection

clip space / normalized device coordinates

⇓ clipping, viewport transform

window space

Input

- vertices in object / model space
- 3D positions
- attributes such as normal, material properties, texture coords

Output

- vertices in window space
- 2D pixel positions
- attributes such as normal, material properties, texture coords
- additional or updated attributes such as
    - normalized depth (distance to the viewer)
    - color (result of the evaluation of the lighting model)

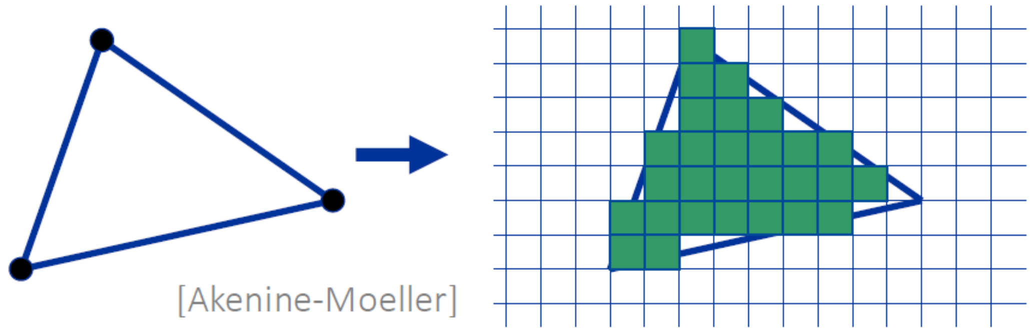
Primitive processing / Rasterization

Input

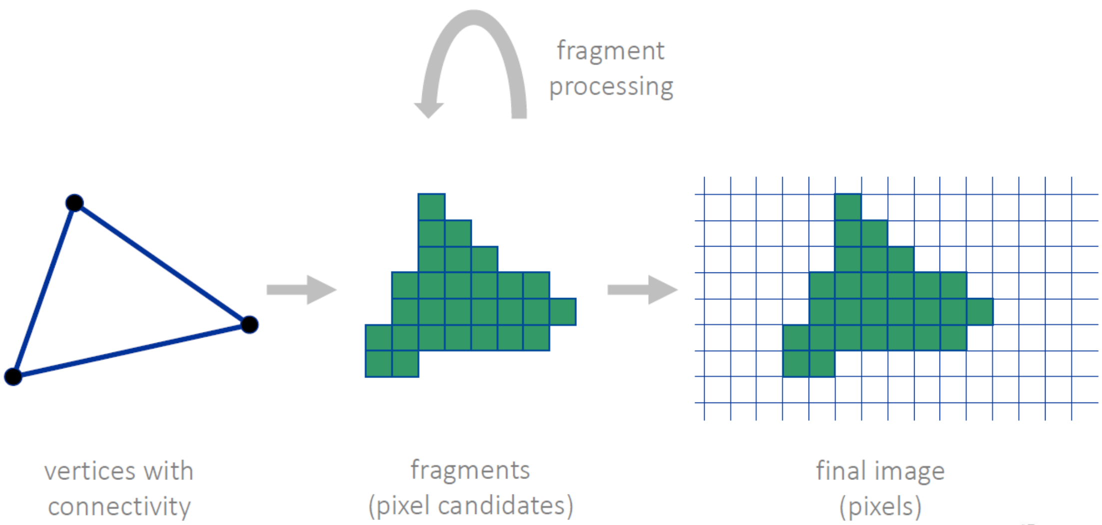
- vertices with attributes and connectivity information
- attributes: color, depth, texture coordinates

Output

- fragments with attributes
- pixel position
- interpolated color, depth, texture coordinates



Fragment processing



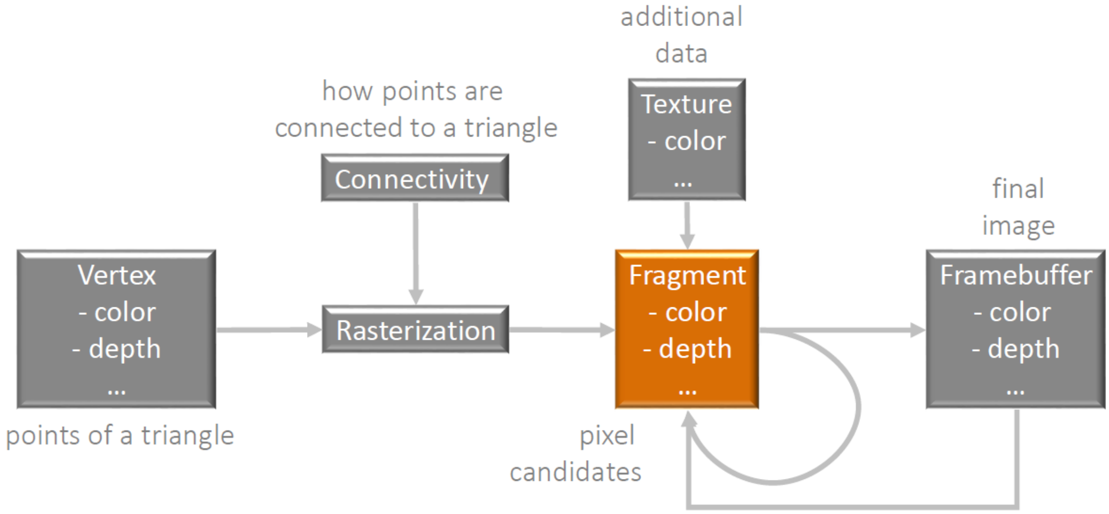
Fragment attributes are processed and tests are performed

- fragment attributes are processed
- fragments are discarded or

- fragments pass a test and finally update the framebuffer

Processing and testing make use of

- fragment attributes
- textures
- framebuffer data that is available for each pixel position
    - depth buffer, color buffer, stencil buffer, accumulation buffer



Attribute Processing

Texture lookup

- use texture cords to look up a texel (pixel of a texture image)

Texturing

- combination of color and texel

Fog

- adaptation of color based on fog color and depth value

Antialiasing

- adaptation of alpha value (and color)
- color has three components: red, green, blue
- color is represented as a 4D vector (red, green, blue, alpha)

Tests

Scissor test

- check if fragment is inside a specified rectangle
- used for, e.g., masked rendering

Alpha test

- check if the alpha value fulfills a certain requirement
- comparison with a specified value
- used for, e.g., transparency and billboarding

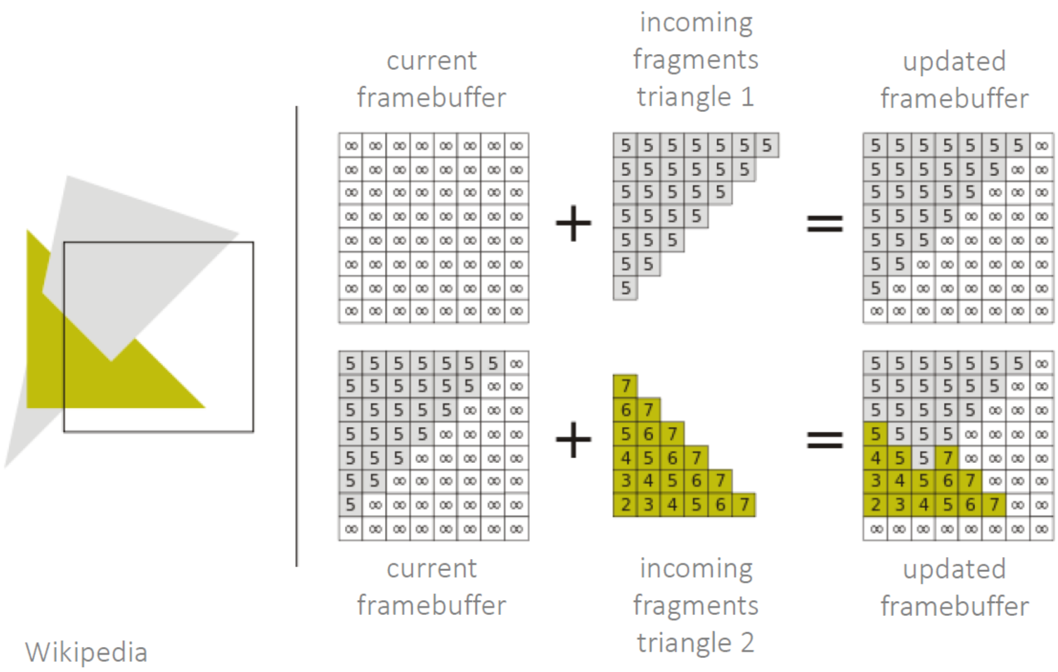
Stencil test

- check if the stencil value in the framebuffer at the position of the fragment fulfils a certain requirement

- comparison with a specified value
- used for various rendering effects, e.g., masking, shadows

Depth Test

- Compare depth value of the fragment and depth value of the framebuffer at the position of the fragment
- Used for resolving the visibility
- If the depth value of the fragment is larger than the framebuffer depth value, the fragment is discarded
- If the depth value of the fragment is smaller than the framebuffer depth value, the fragment passes and (potentially) overwrites the current color and depth values in the framebuffer



Wikipedia

Blending / Merging

Blending

- combines the fragment color with the framebuffercolor at the position of the fragment
- usually determined by the alpha values
- resulting color (including alpha value) is used to update the framebuffer

Dithering

- finite number of colors
- map color value to one of the nearest renderablecolors

Logical operations / masking

Summary

- texture lookup
- texturing
- fog
- antialiasing
- scissor test
- alpha test

- stencil test
- depth test
- blending
- dithering
- logical operations

Rendering Pipeline -Summary

- Primitives consist of vertices
- Vertices have attributes (color, depth, texture coords)
- Vertices are transformed and lit
- Primitives are rasterized into fragments / pixel candidates with interpolated attributes
- Fragments are processed using
  - their attributes such as color, depth, texture coordinates
  - texture data / image data
  - framebufferdata / data per pixel position (color, depth, stencil, accumulation)
- If a fragment passes all tests, it replaces the pixel data in the framebuffer