Computer Graphics
The Standard Graphics Pipeline

Raimond Tunnel
How does the computer work?
How does the computer work?
To the monitor
Computer Graphics

- All about describing the world visuals in math
  - *(Analytical) geometry* – How our objects are shaped?
  - *Linear algebra* – Where our objects are?
From the CPU to the GPU

• On the CPU we specify the data
  • What objects we have (vertices, triangles)
  • What textures they have (images)
  • Where are they in our 3D world (transformations)
  • How are we looking at them (camera, projection)
  • Where are our light sources
  • What is the actual code for the GPU (shaders)
  • ...
  • All the other tricks...
The Standard Graphics Pipeline

- Construct geometry
  - Define transformations
  - Assign material properties
  - ... 

- Vertex Transformations
- Culling & Clipping
  - Determine front-facing triangles
  - Determine which vertices are visible

- Rasterization
  - Fill the triangle with fragments

- Fragment Shading
  - Calculate correct color values

- Visibility Tests
  - Blending
    - Is the fragment visible?
    - Blend together multiple fragments

- Frame buffer
The Standard Graphics Pipeline

- Construct geometry
  - Define transformations
  - Assign material properties

- Vertex Transformations

- Culling & Clipping
  - Determine front-facing triangles
  - Determine which vertices are visible

- Rasterization
  - Fill the triangle with fragments

- Fragment Shading
  - Calculate correct color values

- Fragment Shader
  - Is the fragment visible?
  - Blend together multiple fragments

- Visibility Tests

- Blending

- Pipeline output
Dear GPU, ...
1. Vertex Transformations

Matrices:

- **Model** – Where is our object?
- **View** – Where is our camera?
- **Projection** – How to get on the screen?

All vertex coordinates are transformed to clip space and then to screen space.
1. Vertex Transformations
Every vertex is transformed into the world's coordinates.
1. Vertex Transformations

\( \mathbf{V} \) transforms every vertex into camera's coordinates.

\( \mathbf{P} \) does the projection from 3D to 2D (roughly speaking).
1. Vertex Transformations

- In the end all the vertices get **screen coordinates**!
- There are a couple of other steps there too...
1. Vertex Transformations

- Most of the transformation is in a *shader*
- *Shader* is a piece of code sent to the GPU
- There are different shaders, which get executed in different steps

- **Vertex shader** – Executed on each vertex
  Aim is to transform the vertex coordinates from the model's local space to clip space

- GPU does other things automatically to get to NDC-space and finally to screen space later...
1. Vertex Transformations

This shader code will be executed on each vertex.

Although it is called vertex shader, it does not actually shade anything...
The Standard Graphics Pipeline

- Construct geometry
- Define transformations
- Assign material properties

**Vertex Transformations**

**Vertex Shader**
Object's local space → viewport space

- Culling & Clipping
  - Determine front-facing triangles
  - Determine which vertices are visible

- Rasterization
  - Fill the triangle with fragments

**Fragment Shading**
Calculate correct color values

- Visibility Tests
- Blending
  - Is the fragment visible?
  - Blend together multiple fragments
2. Culling & Clipping

- 3D meshes are made up of **triangles**
- Triangle has:
  - 3 vertices
  - 3 edges
  - 1 two-sided face

In a right-handed coordinate system the **positive** angular direction is **counter-clockwise**!
2. Culling & Clipping

- In step 3 all the triangles will be considered...

- But usually we do not care about the faces inside of the mesh.
2. Culling & Clipping

- **Back-face culling** ignores all the back faces of the triangles.
- Those faces will not be sent to the later steps.
2. Culling & Clipping

- Triangles completely outside the camera's view space are also ignored
2. Culling & Clipping

- It may happen that some triangle is partially in view:
2. Culling & Clipping

- These triangles will be **clipped**

Viewport (the screen)

But wait! This is not a triangle!
2. Culling & Clipping

- These triangles will be clipped

Viewport (the screen)
The Standard Graphics Pipeline

- Construct geometry
- Define transformations
- Assign material properties

- Vertex Transformations

- Culling & Clipping
  - Determine front-facing triangles
  - Determine which vertices are visible

- Rasterization
  - Fill the triangle with fragments

- Vertex Shader
  - Object's local space → viewport space

- Fragment Shading
  - Calculate correct color values

- Visibility Tests
  - Is the fragment visible?
- Blending
  - Blend together multiple fragments
3. Rasterization

- So far we were in an ideal mathematical world...
- Time for a reality check!
3. Rasterization

- Our screen is a discrete **grid of pixels**!
- Rasterization is a process of finding out which triangles occupy which pixels.
3. Rasterization

- This is a source of **aliasing** in computer graphics
- Aliasing – creating something unintentional
3. Rasterization

- Rasterization creates *rasters / fragments*
- Those are passed to step 4. fragment shading
- Vertices can also have *attributes*, which have their values *interpolated* to the fragments
3. Rasterization

---

I am blue!

---

I am red!

---

I am blue!

---

We are all in between!

---

I am red!
3. Rasterization

- Interpolation is done by the GPU
- Also works on triangles

Common interpolated values:
- Color
- Position
- Normal vector
- UV coordinates
The Standard Graphics Pipeline

- Construct geometry
  - Define transformations
  - Assign material properties

- Vertex Transformations

- Culling & Clipping
  - Determine front-facing triangles
  - Determine which vertices are visible

- Rasterization
  - Fill the triangle with fragments

- Fragment Shading
  - Calculate correct color values

- Visibility Tests
- Blending
  - Is the fragment visible?
  - Blend together multiple fragments

- Display
4. Fragment Shading

- The **fragment shader** is another piece of code, which is ran on each of the fragments!
- Purpose is to calculate the **color** for a fragment!

[Image of a red hexagon on a black background]

http://cgdemos.tume-maailm.pri.ee/
4. Fragment Shading

• What is color?
4. Fragment Shading

- There are many models for calculating the reflected light
- Some are more physically correct, some are not
- They are all written and executed in the f. shader
4. Fragment Shading

- We usually need to know for each fragment:
  - Position (coordinates) of the fragment in the world
  - The surface normal at the fragment's location
  - UV coordinates or color information
  - The position(s) of the light source(s) in our world
  - The position of the camera in our world

- We will see this in more detail next time!

Promise! ;)

The Standard Graphics Pipeline

1. **Construct geometry**
   - Define transformations
   - Assign material properties

2. **Vertex Transformations**

3. **Culling & Clipping**
   - Determine front-facing triangles
   - Determine which vertices are visible

4. **Rasterization**
   - Fill the triangle with fragments

5. **Fragment Shading**
   - Calculate correct color values

6. **Visibility Tests**
   - Is the fragment visible?
   - Blend together multiple fragments
5. Visibility Tests & Blending

- How does the GPU know, which object is in front?
- How to handle overlapping objects?
5. Visibility Tests & Blending

- Besides the screen coordinates, each fragment also has a **depth value**.
- When drawing each fragment, the depth value is checked against a previous depth value of a fragment drawn on the same coordinates.

I am from triangle A
I need to cover this pixel
My depth value is 0.8
0.8 < 1 → **OK**
5. Visibility Tests & Blending

I am from triangle B
I need to cover this pixel
My depth value is 0.2
0.2 < 0.8 → OK

I am from triangle C
I need to cover this pixel
My depth value is 0.5
0.5 > 0.2 → NO
5. Visibility Tests & Blending

• What about transparent objects (glass etc)?
5. Visibility Tests & Blending

- Transparent objects are rendered after opaque
- Transparent objects are ordered by distance
- Colors are **blended together**
5. Visibility Tests & Blending

- Colors can be blended together in many ways:
  - **Alpha-blending** – your regular semi-transparency
  - **Additive blending** – colors are added (like light)
  - **Multiplicative blending** – colors multiply (like glaze paint)

http://www.andersriggelsen.dk/glblendfunc.php
5. Visibility Tests & Blending

- Different blending create different visual effects
- GPU allows you to configure the modes

*World Remade* by Jaanus Jaggo
The Standard Graphics Pipeline

1. Construct geometry
   Define transformations
   Assign material properties...

2. Vertex Transformations

3. Culling & Clipping
   Determine front-facing triangles
   Determine which vertices are visible

4. Rasterization
   Fill the triangle with fragments

5. Fragment Shading
   Calculate correct color values

6. Visibility Tests Blending
   Is the fragment visible?
   Blend together multiple fragments

7. Display on monitor
The Standard Graphics Pipeline

- Now you have a good knowledge of the GPU pipeline!
- Feel free to check out the CG material here: https://cglearn.codelight.eu/pub

- Next time we will try to program some shaders!

Thank you!