Next-Gen Games means Next-Gen Graphics

Karol Gasiński

Graphic Software Engineer

November, 18 2013

VPG Graphics Software Development & Validation

Abstract

This presentation will guide you through real time computer graphics, based on examples from AAA game titles, and Next-Gen engines, that are currently in development. We will start by explaining asset and rendering pipelines by showing their usage in creation of different effects. During that process basics of OpenGL will be introduced, as it is rendering API that makes it all possible.

Some content presented in this slides is a result of my activities performed outside Intel, out of working hours, and without use of any Intel assets. It is not influencing my work and cannot be treated as a concurrency to Intel business. All information showed here represents my personal believes and don't represent position of Intel Corporation.

This presentation was based on my earlier lectures given during different conferences and events. Therefore similar content can be found in web.

Agenda

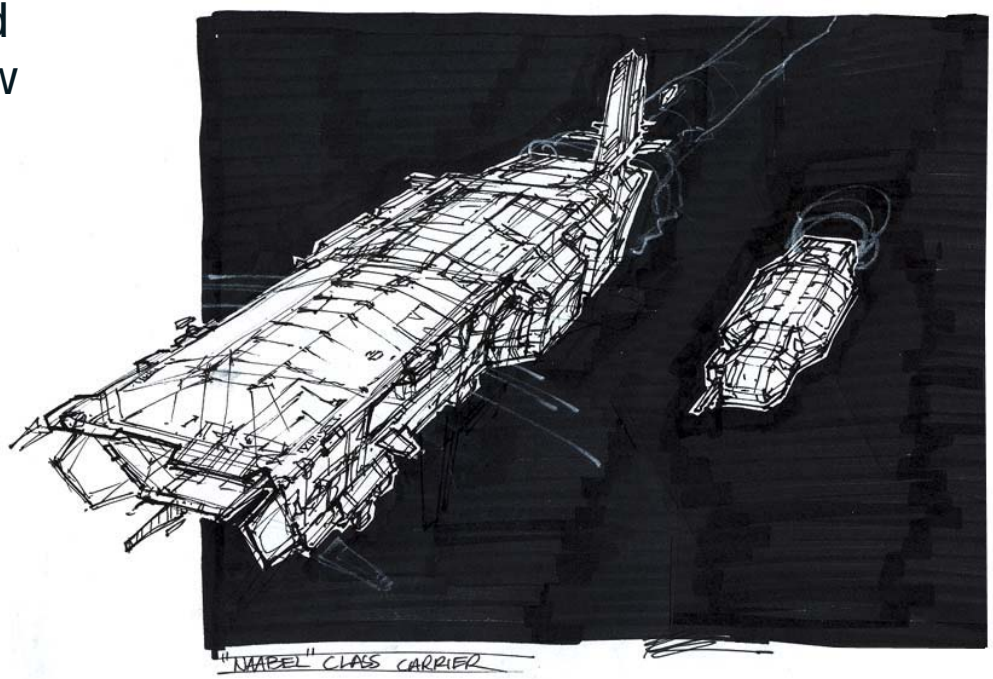
- Asset Pipeline
  - Concept Art
  - Modeling
  - Texturing
- Rendering Pipeline
  - Vertex Shader
  - Tessellation Shaders
  - Geometry Shader
  - Fragment Shader
  - Compute Shader
- Conclusion

Asset Pipeline

Asset Pipeline and Creation

Concept Art

- To create initial design and shape, concept arts for few projections and situational drafts are made.
- For humans and creatures this is few poses in action.
- Traditional way was paper, pencils, markers

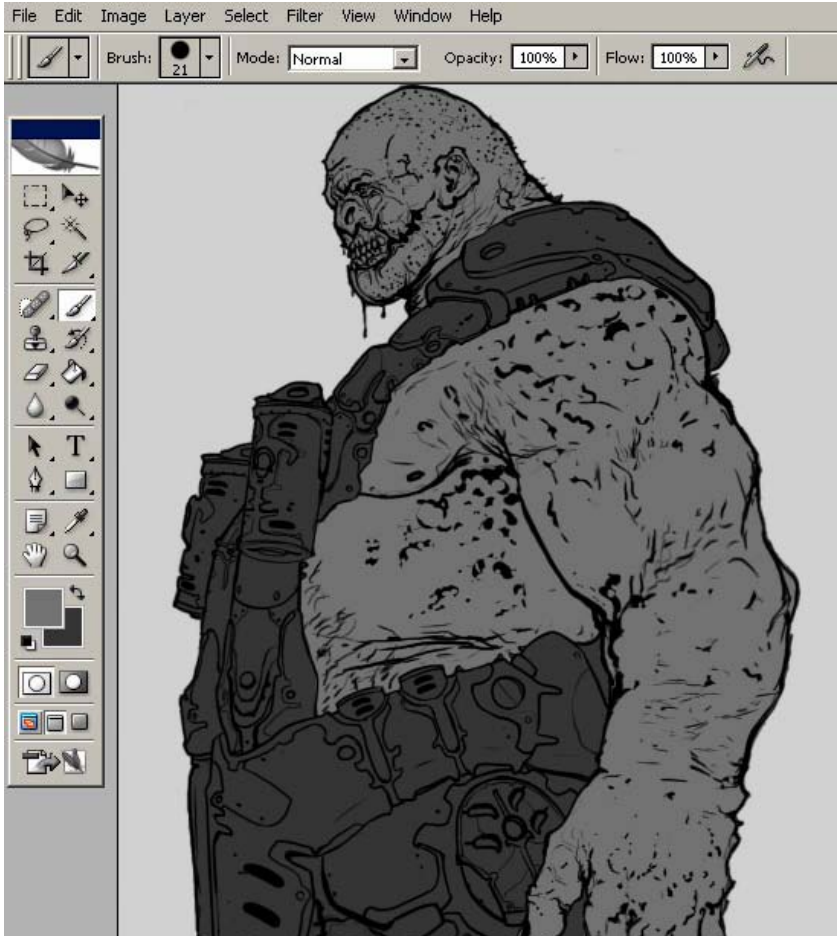


Homeworld Concept Art (~1997)

Asset Pipeline and Creation

Concept Art

- Currently all content and prototyping is done in software which simplifies whole process.
- On right, concept art of “Rabagh The Bombardier” done by Vitaly Bulgarow in Photoshop.
- We will use his work as an example of next-gen asset creation.
- Whole tutorial can be bought here.



Asset Pipeline and Creation

Mass Effect 2 Concept Art done using Wacom tablet with underlying display.

Asset Pipeline and Creation

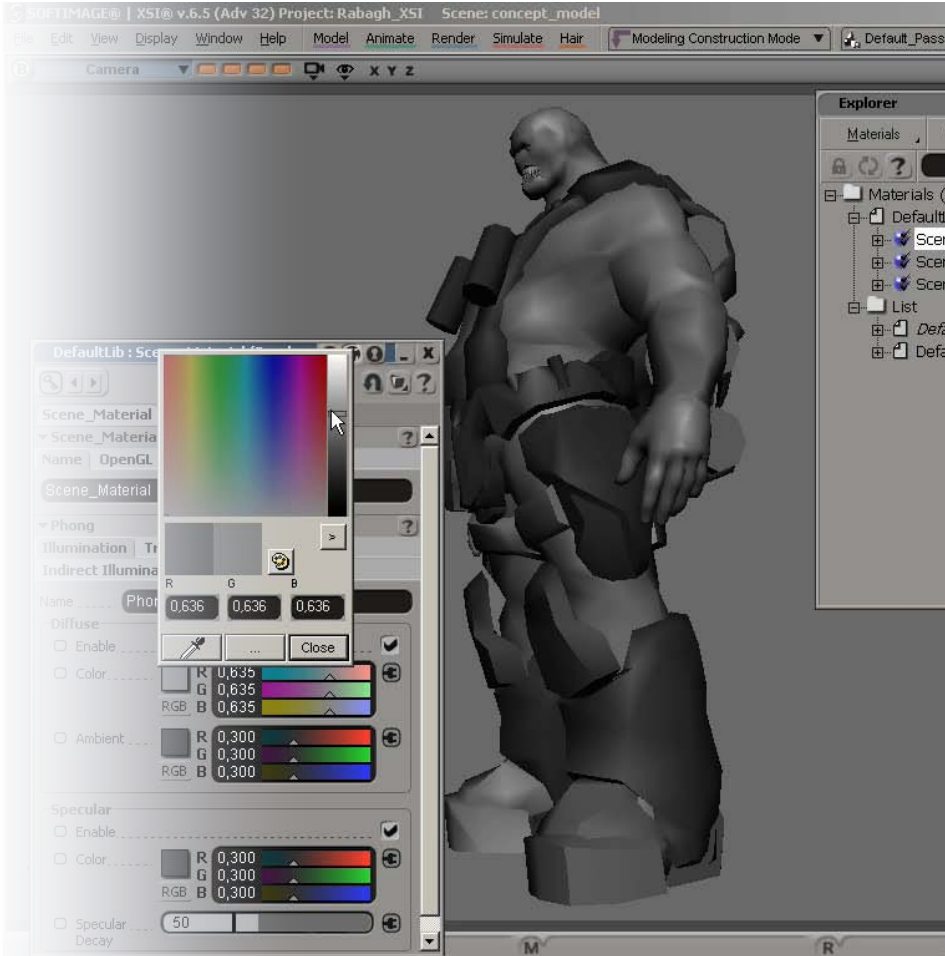
Concept Art

- Final concept art is ready. Now it is time to create model based on it.
- Concept arts can be used directly during modeling as a background for reference.
- They can be also set as a three projections for each of the axes, which is useful during creation of vehicles and buildings.

Asset Pipeline and Creation

Modeling

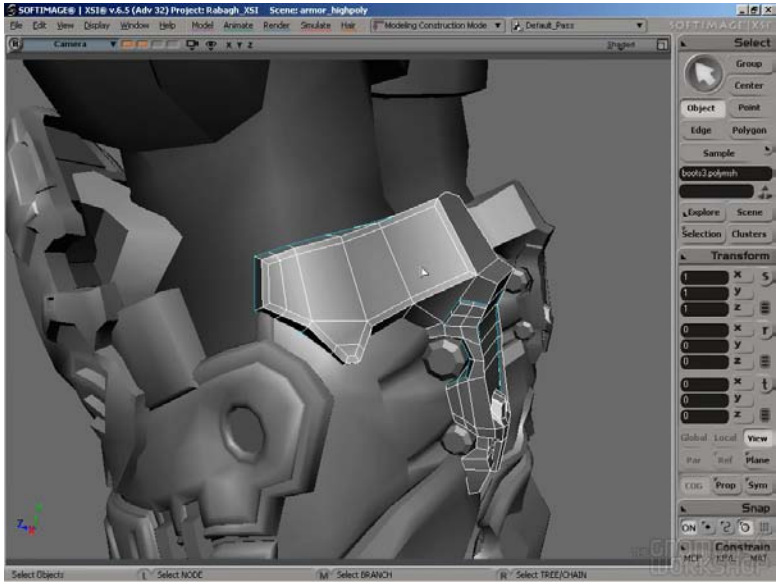
- Base human model with animation skeleton is used as a starting point.
- It is divided into parts that will have different materials.
- Each part is modeled separately to get the final outcome.
- Here initial phase of “Rabagh” in Softimage XSI 6.5



Asset Pipeline and Creation

Modeling

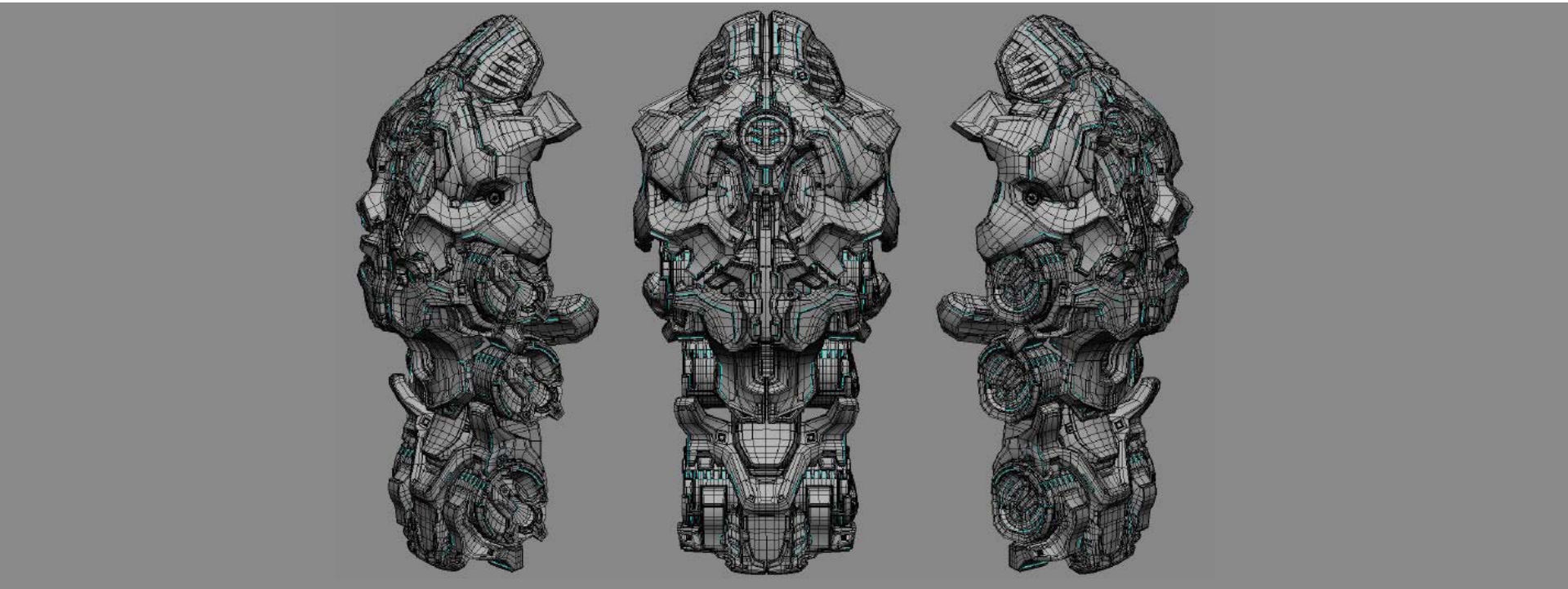
- Iterating step by step from low poly, high poly model is created with all the final detail.



Asset Pipeline and Creation

Modeling

- Technical parts with hard surfaces like this backpack were modeled in Softimage. Here in wireframe mode.



Asset Pipeline and Creation

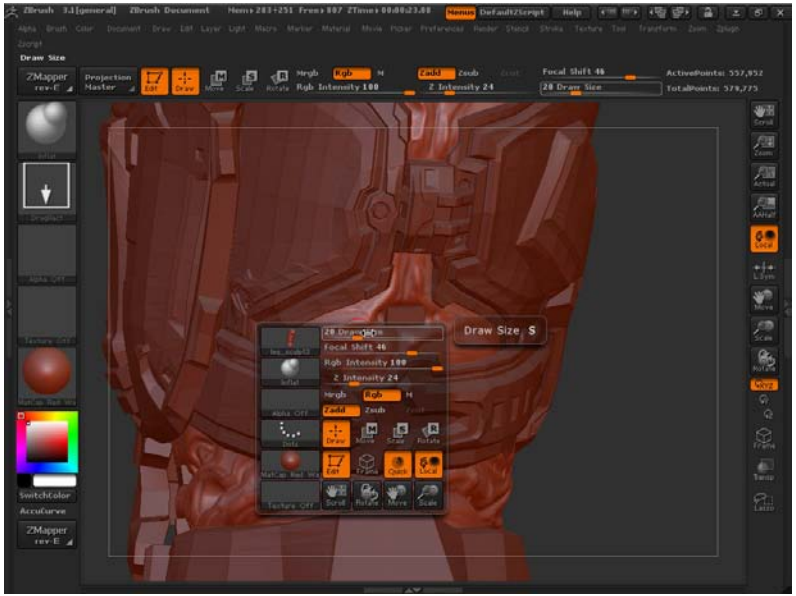
Modeling

- Final high poly model of backpack presented using offline rendering.

Asset Pipeline and Creation

Modeling

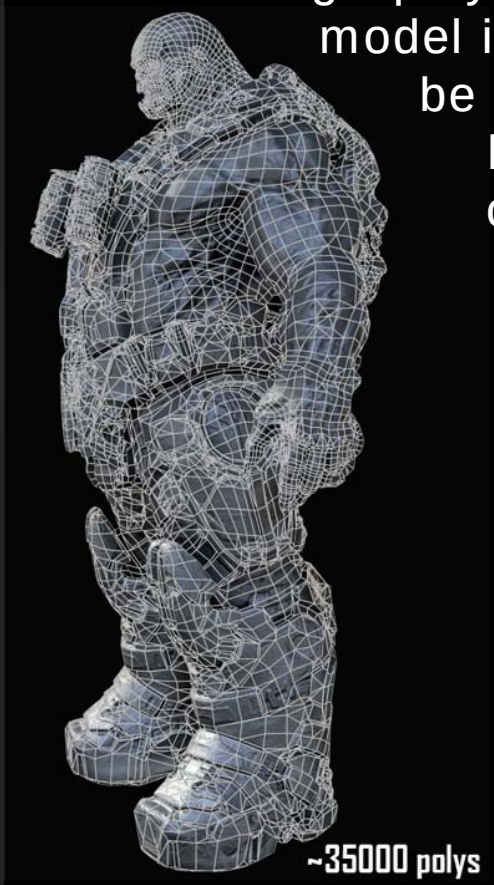
- Sculpting tool like ZBrush is ideal for modeling high-poly organic parts (it's like forming clay).



Asset Pipeline and Creation

High-poly model is ready. It is build from few millions of polygons. It is definitely too much geometry for any graphic card to render in real time.

Based on high-poly version, low-poly model is generated. It can be done automatically by tools, and then corrected manually by the artist.



Here you can see 35K poly version of model.

It will be used to render enemies located very close to the player.

Asset Pipeline and Creation

Modeling

- When required polygon budget is reached during generation of low-poly model, both versions are ready. Now it is time to create textures.

Asset Pipeline and Creation

Texturing

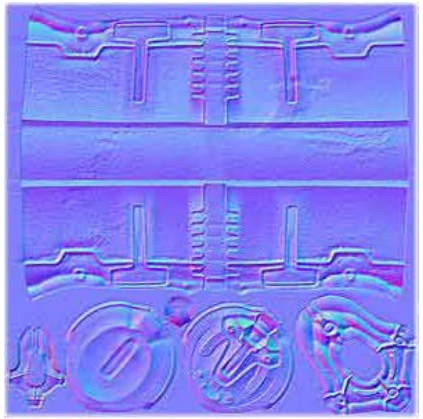
- Each low-poly part of model is now sliced and projected on 2D surface.
- As a result each point of model gets coordinates in 2D space of texture.
- This is called unwrapping and allows creation of textures that will be used with model.
- On right, three 2048x2048 already painted textures for legs, belt and organic surfaces.

Asset Pipeline and Creation

Texturing

- Using difference in shape between high-poly and low-poly model, it is possible to automatically generate displacement maps (called also height maps).
- From them also normal maps are generated.
- Displacement, height and normal maps will be explained further in this presentation.
- Two 1024x1024 textures with their normal maps.



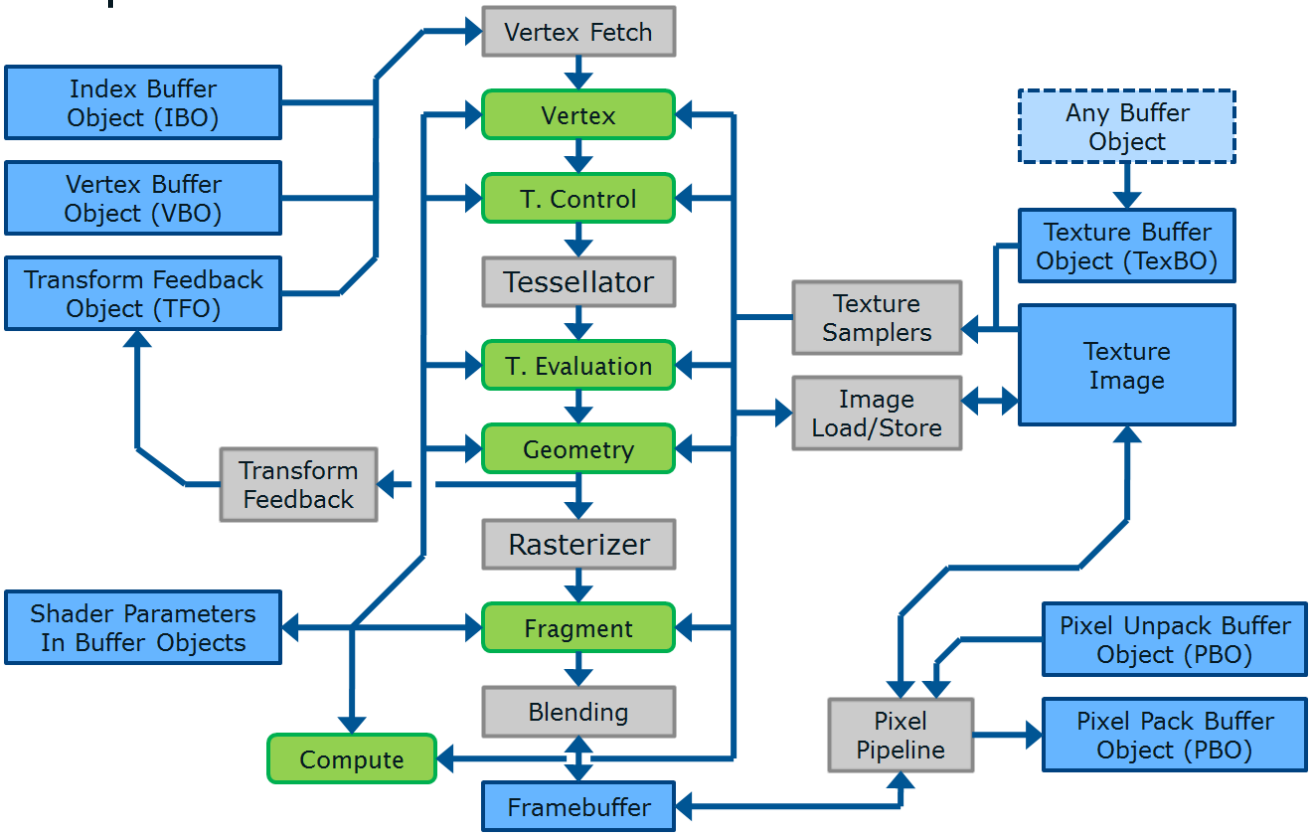
Asset Pipeline and Creation

Final model with full set of textures is ready. To be used in game it still needs whole set of different animations to be created like walking, crouching, jumping etc. Technical Artist will also need to create final materials for it, using prepared diffuse, specular, normal and displacement textures.

Rendering Pipeline

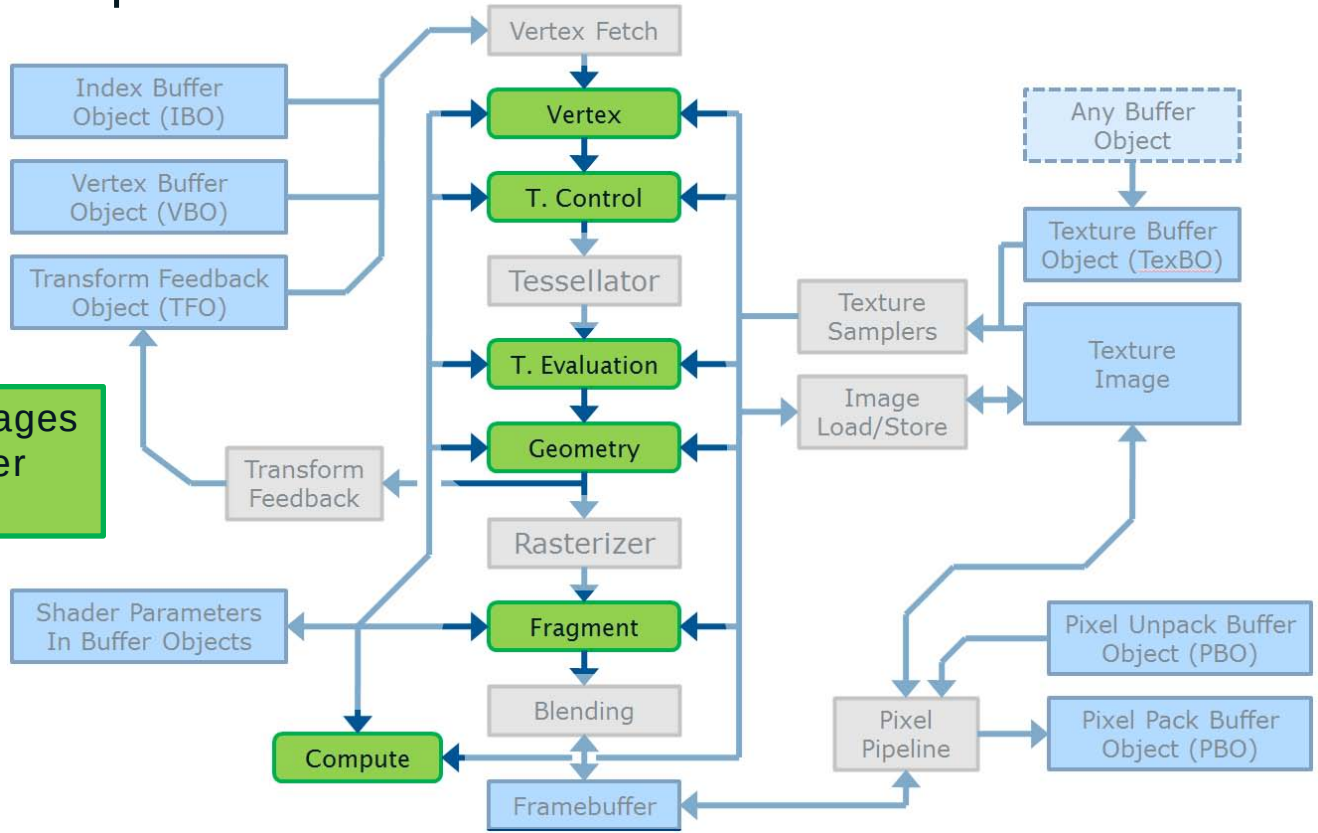
Rendering Pipeline

OpenGL 4.4 Pipeline:



Rendering Pipeline

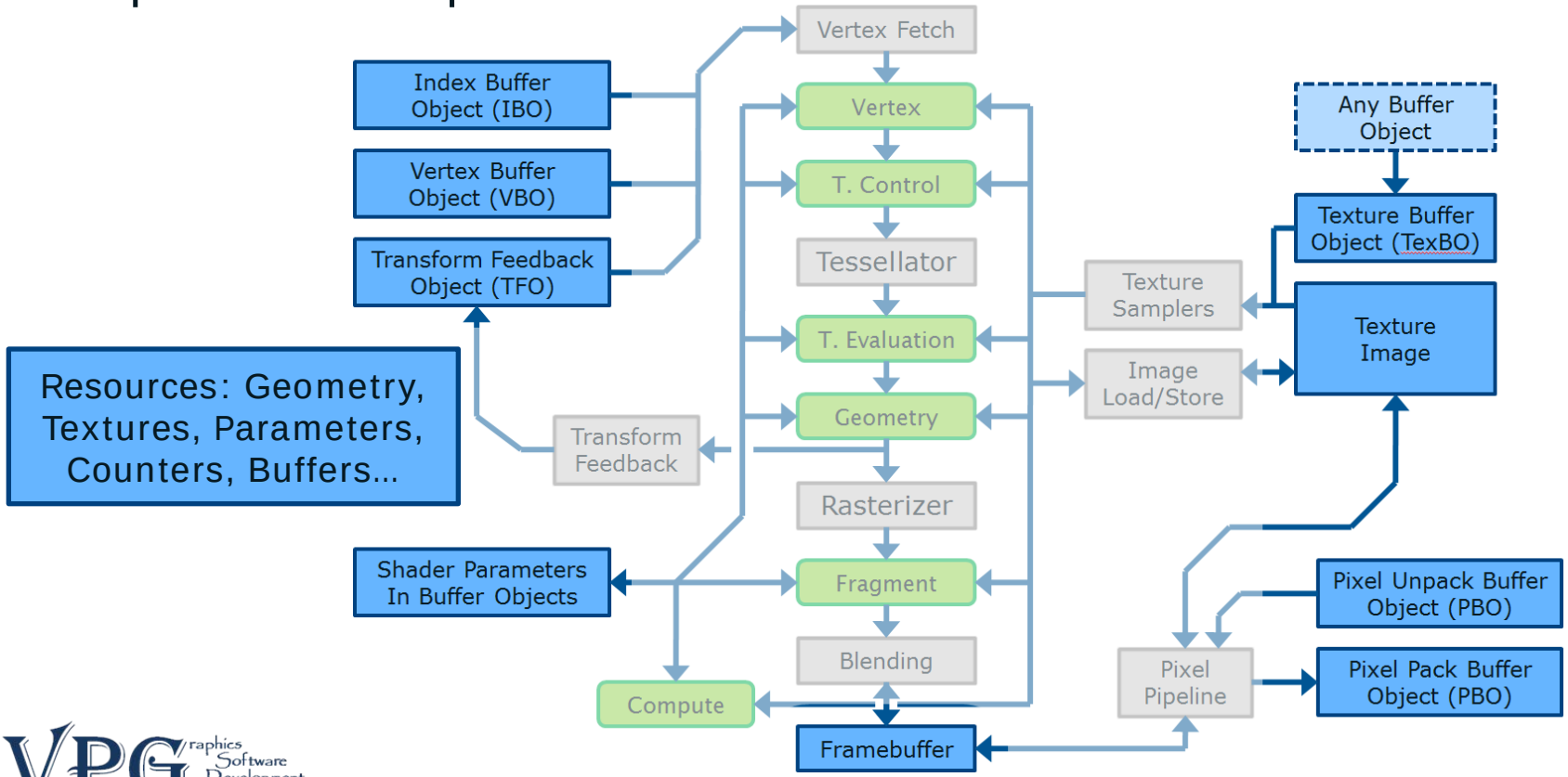
OpenGL 4.4 Pipeline:



Programmable Stages Executes Shader Programs

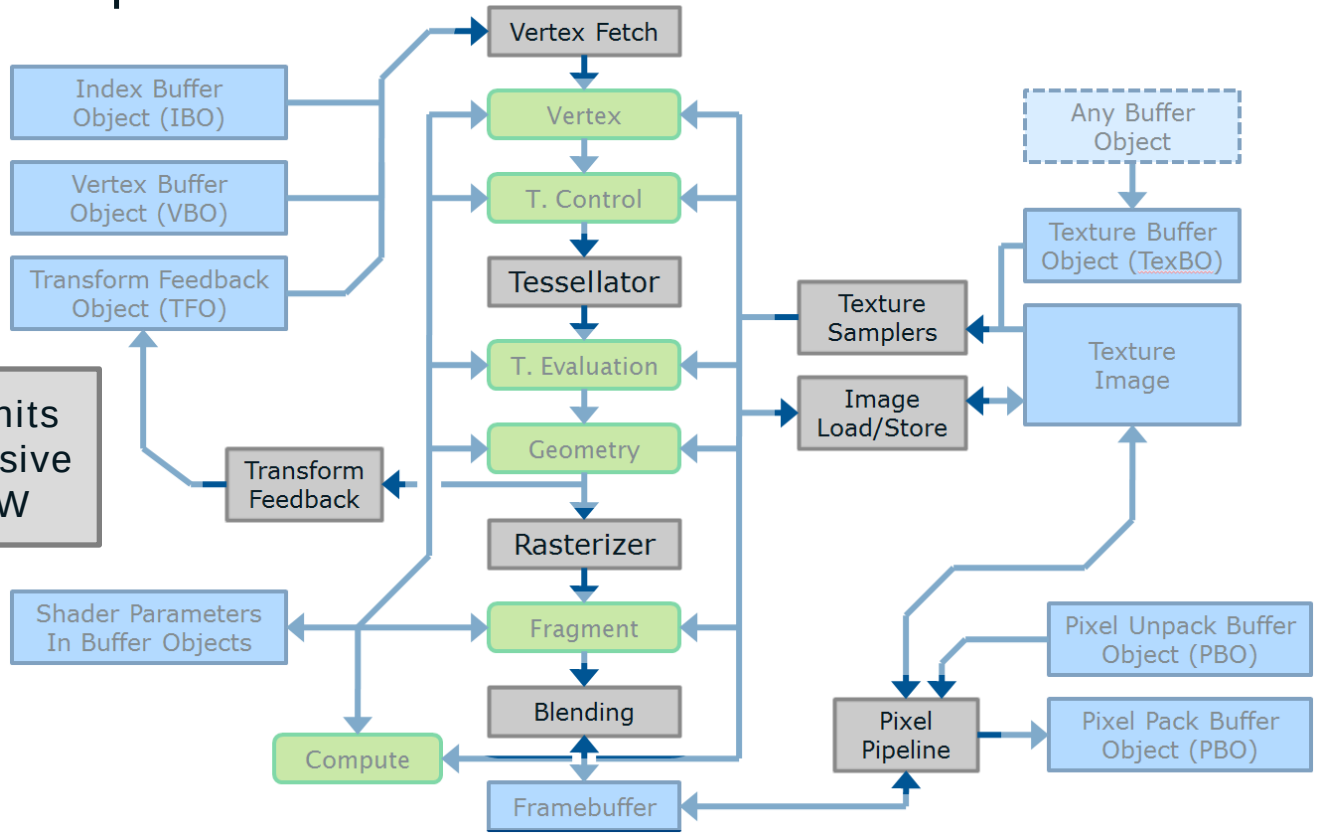
Rendering Pipeline

OpenGL 4.4 Pipeline:



Rendering Pipeline

OpenGL 4.4 Pipeline:

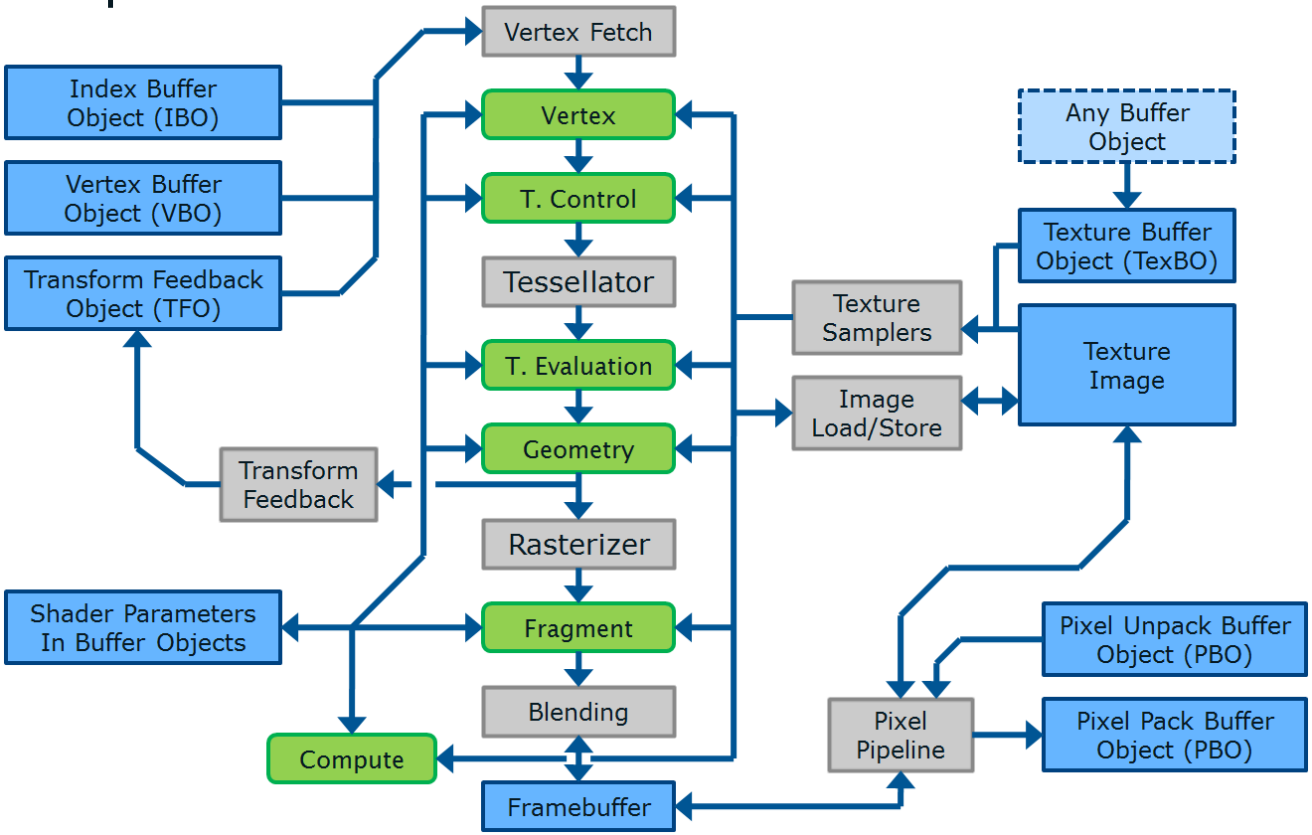


Fixed Function Units performing expensive operations in HW

Vertex Shader

Rendering Pipeline

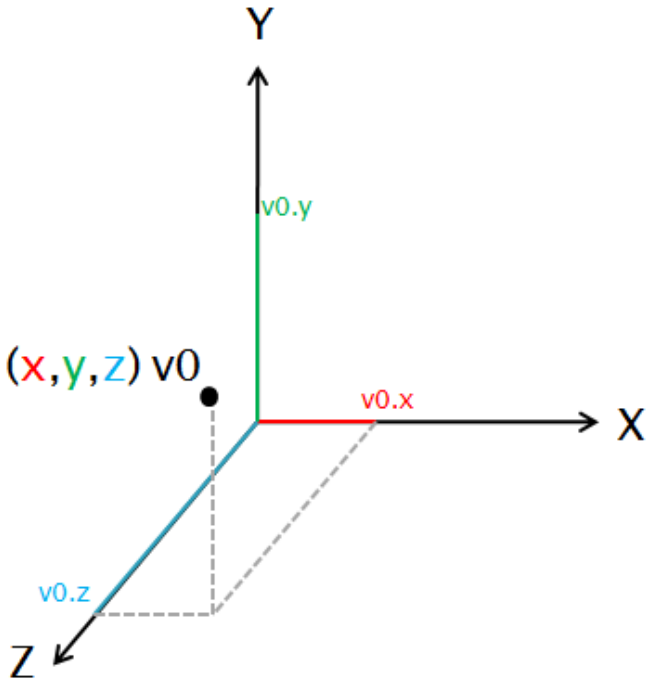
OpenGL 4.4 Pipeline:



Graphic Primitives

Vertex (*pol. wierzchołek*)

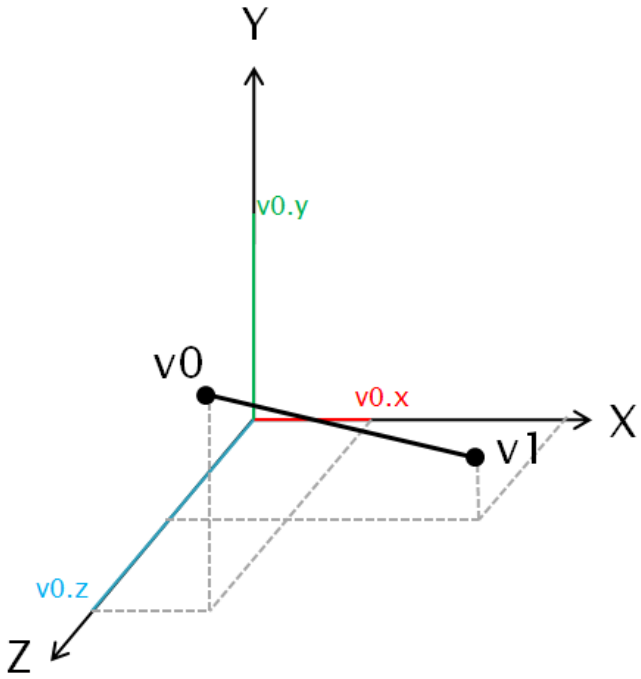
- has position in 3D space
- can have other attributes
  - color, texture coordinates, animation weights, etc.
- is a base primitive used to describe all other types of geometry primitives that can be processed by graphic cards



Graphic Primitives

Line (*pol. linia*)

- build from two vertexes
- mostly used in specialized CAD software for displaying wireframes of designs
- in games used for rendering hairs, vector graphics in very old games, etc.



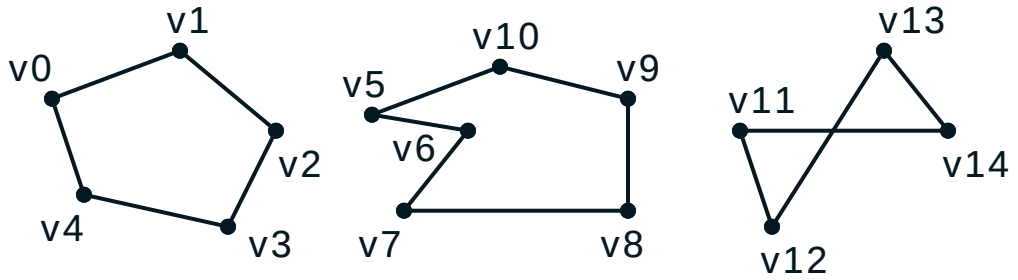
Graphic Primitives

Vertex and Line types accepted by OpenGL:

Points
(GL_POINTS)

Separate Lines
(GL_LINES)

Line Loops
(GL_LINE_LOOP)



Graphic Primitives

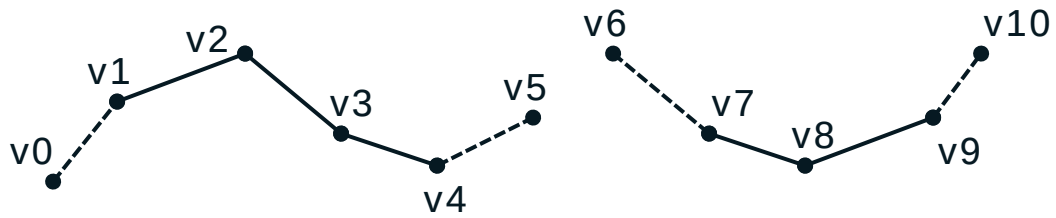
Vertex and Line types accepted by OpenGL:

Line Stripes
(GL_LINE_STRIP)

Special Types*:

Separate Lines
with Adjacency
(GL_LINES_ADJACENCY)

Line Stripes
with Adjacency
(GL_LINE_STRIP_ADJACENCY)

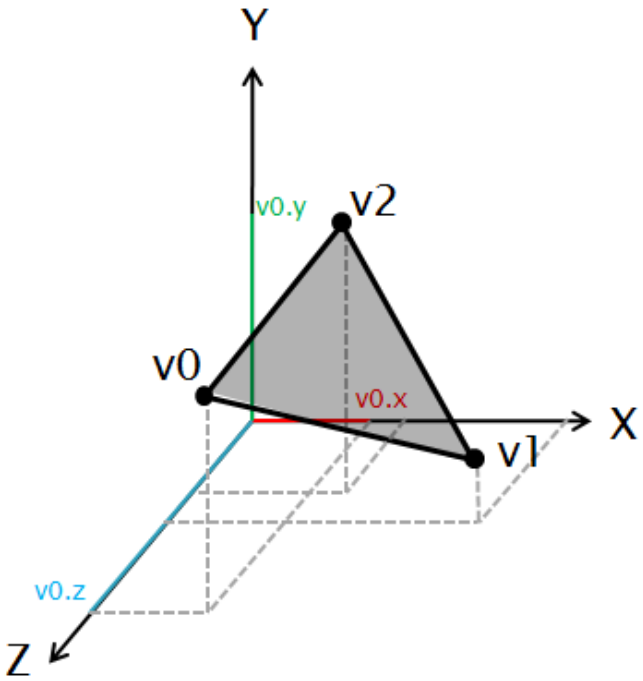


*Will be explained with Geometry Shaders.

Graphic Primitives

Triangle

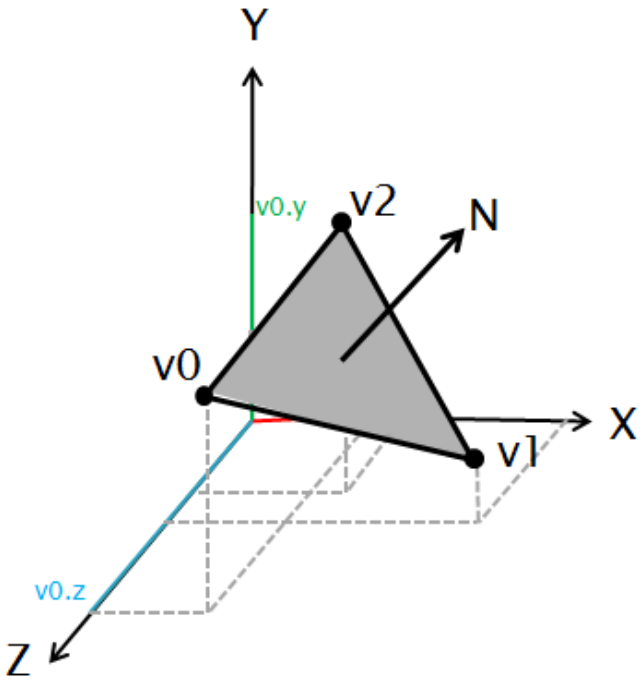
- simplest primitive capable of describing a surface
- most important primitive in computer graphics
- all three dimensional forms are build from triangles
- used also in 2D graphics to describe User Interfaces, sprites, etc.



Graphic Primitives

Normal vector

- vector perpendicular to triangle surface
- describes towards which direction triangle is facing, which side of it is visible

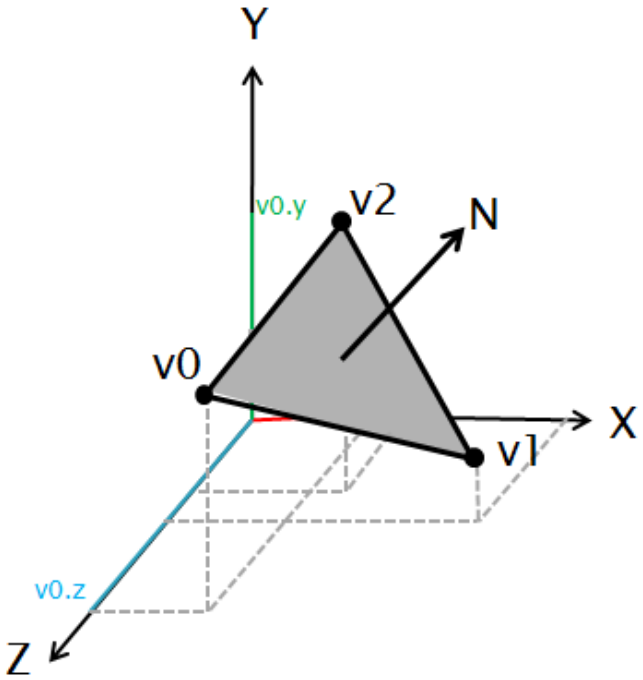


Graphic Primitives

Normal vector

Used for:

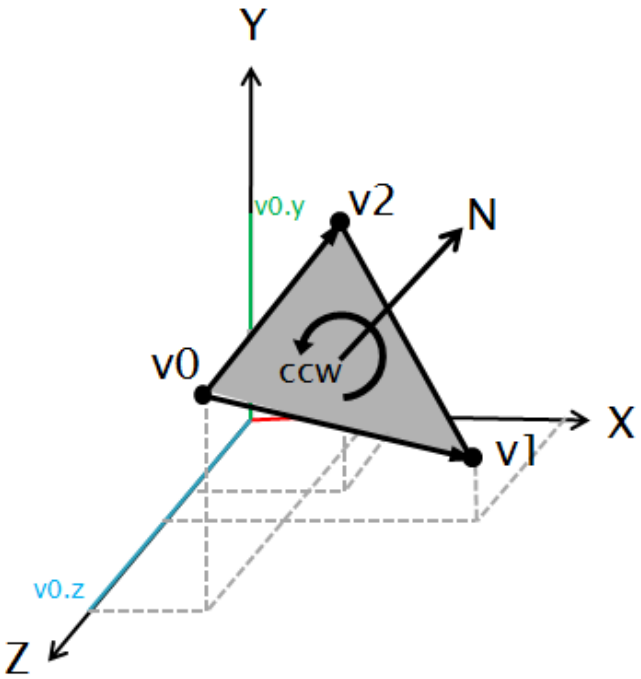
- visibility check (back face culling)
  each triangle has two sides: visible and invisible,
  when normal is pointing towards observer - it's visible
- surface lightning properties
  needed for light calculation - determine how much light is
- tessellation calculations
  normal vectors are used during displacement of
  vertices generated by Tessellator
- distance calculation
  dot product of normal and point returns distance
  from surface defined by triangle to this point



Graphic Primitives

Normal vector

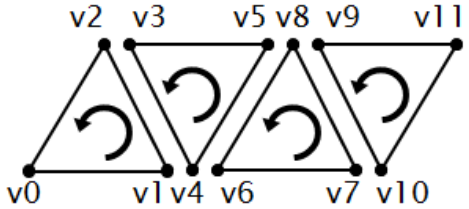
- calculated using cross product, possible two directions of calculation:
  - right hand rule
    CCW (Counter Clockwise)
    N = (v1 - v0) x (v2 – v0)
  - left hand rule
    CW (Clockwise)
    N’ = (v2 - v0) x (v1 – v0)
  - N = -N’



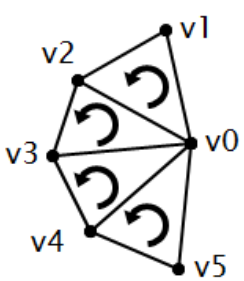
Graphic Primitives

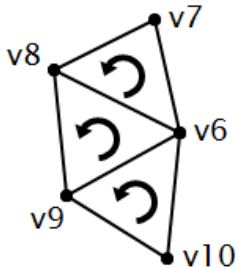
Triangle types accepted by OpenGL:

Separate Triangles
(GL_TRIANGLES)

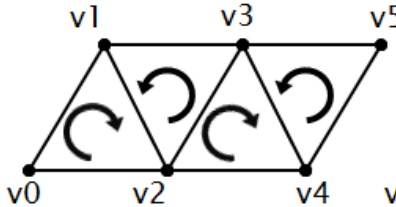


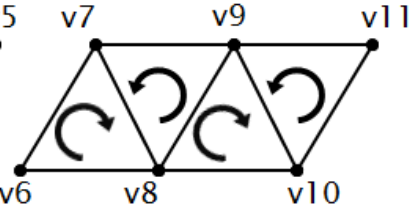
Triangle Fans
(GL_TRIANGLE_FAN)





Triangle Stripes
(GL_TRIANGLE_STRIP)

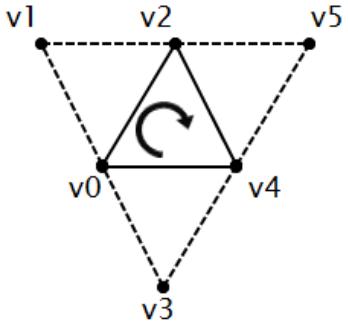




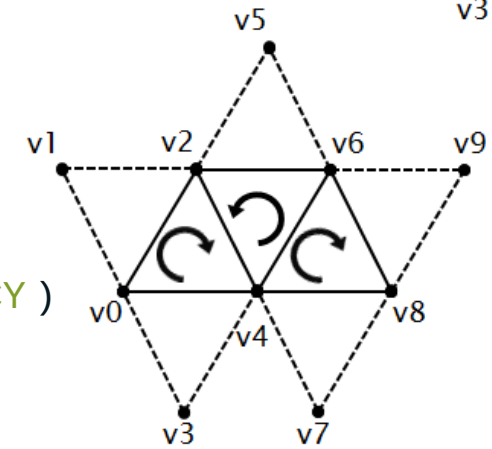
Graphic Primitives

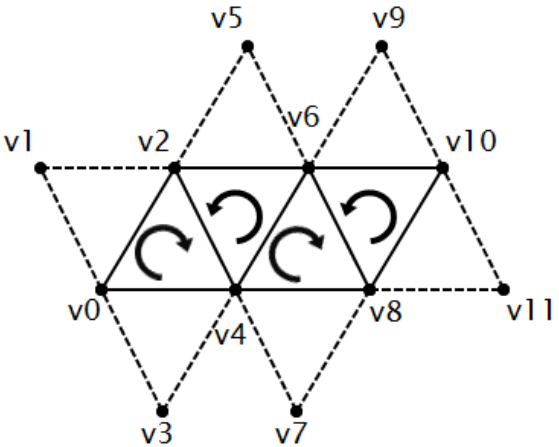
*Special Triangle types accepted by OpenGL:

Separate Triangles
with Adjacency
(GL_TRIANGLES_ADJACENCY)



Triangle Stripes
with Adjacency
(GL_TRIANGLE_STRIP_ADJACENCY)

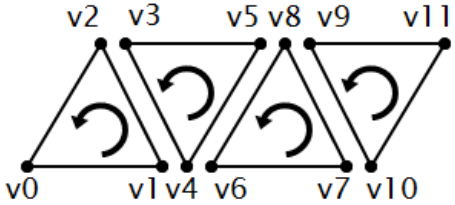




*Will be explained with Geometry Shaders.

Graphic Primitives

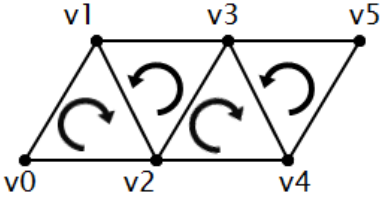
Separate Triangles



- Most 3D models are closed.
- This means each triangle has surrounding neighbor triangles with which it shares its vertexes.
- It would be a waste of VRAM, bandwidth and compute power to send the same vertex N times for each triangle that uses it.
- Specialized programs can generate triangle stripes.

vs

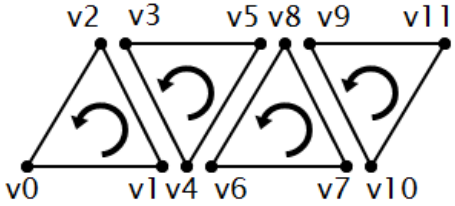
Triangle Stripes



- In Triangle Stripe each triangle is created from current vertex and previous two.
- This causes vertex order to be reversed every second triangle, which means their normal vectors would be wrongly calculated.
- During rendering GPU takes it into notice to preserve normals.

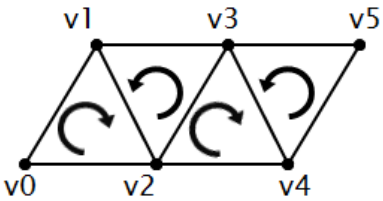
Graphic Primitives

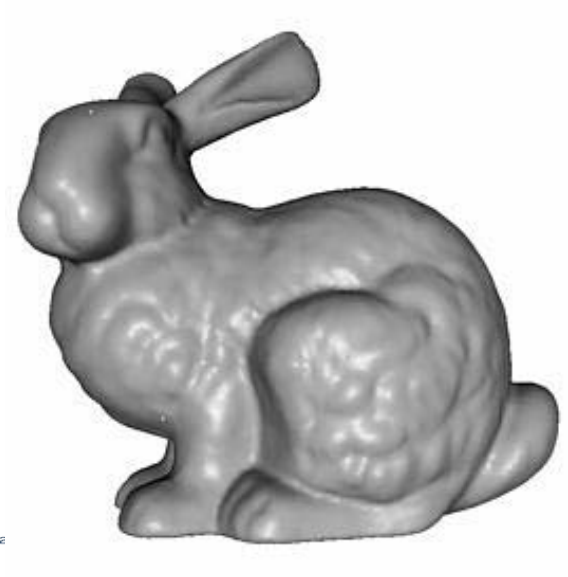
Separate Triangles



vs

Triangle Stripes





Vertex Shader

Explosions, gas, smoke

- Animation and movement of big amount of simple particles with given lifecycle and emitters

Foliage and plants

- Animation of grass, leaves and small plants based on force of wind and it's direction
- Can also take into notice for eg. Surrounding explosions

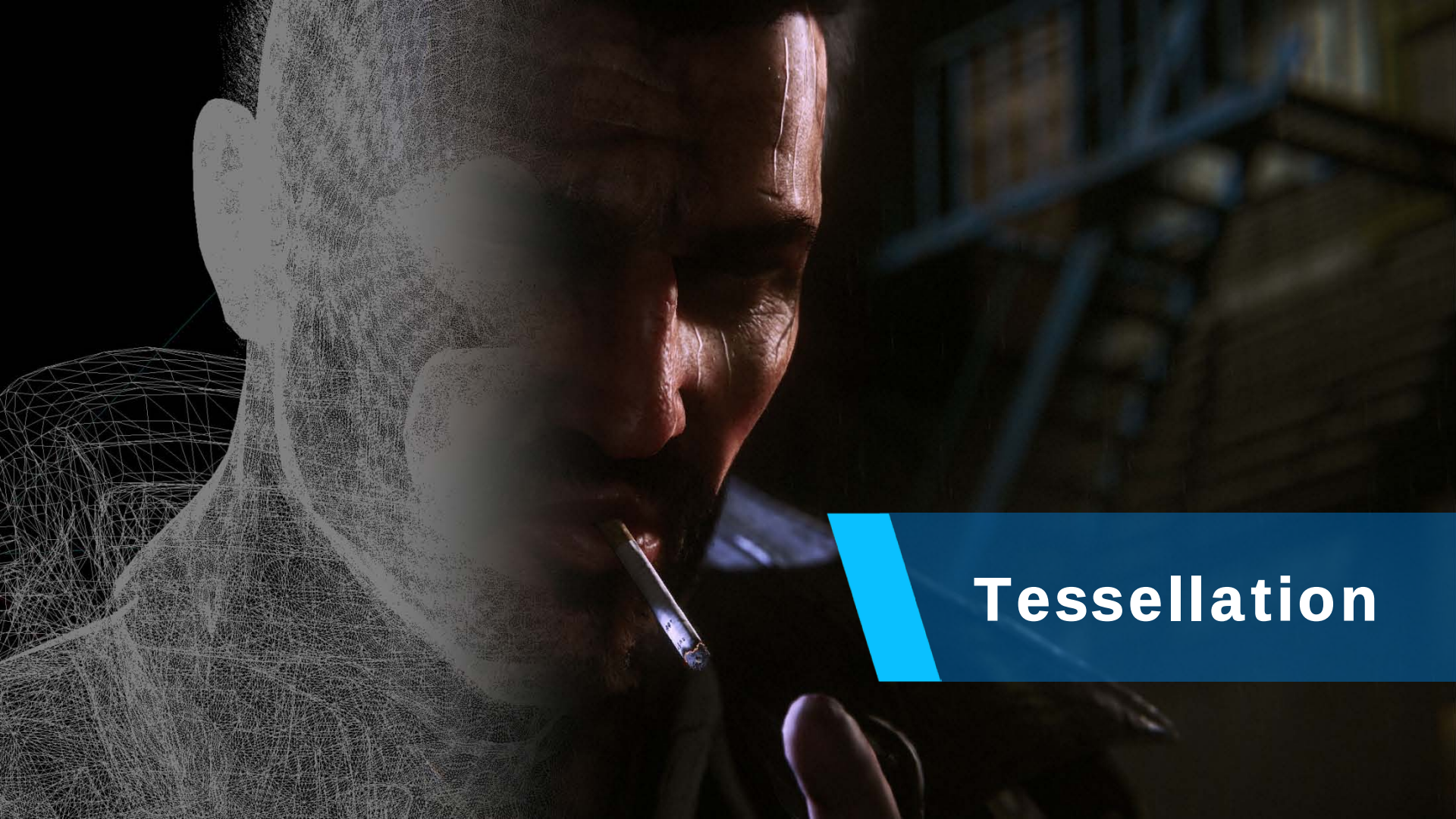
Vertex Shader

Explosions, gas, smoke

- Animation and movement of big amount of simple particles with given lifecycle and emitters

Foliage and plants

- Animation of grass, leaves and small plants based on force of wind and it’s direction
- Can also take into notice for eg. Surrounding explosions

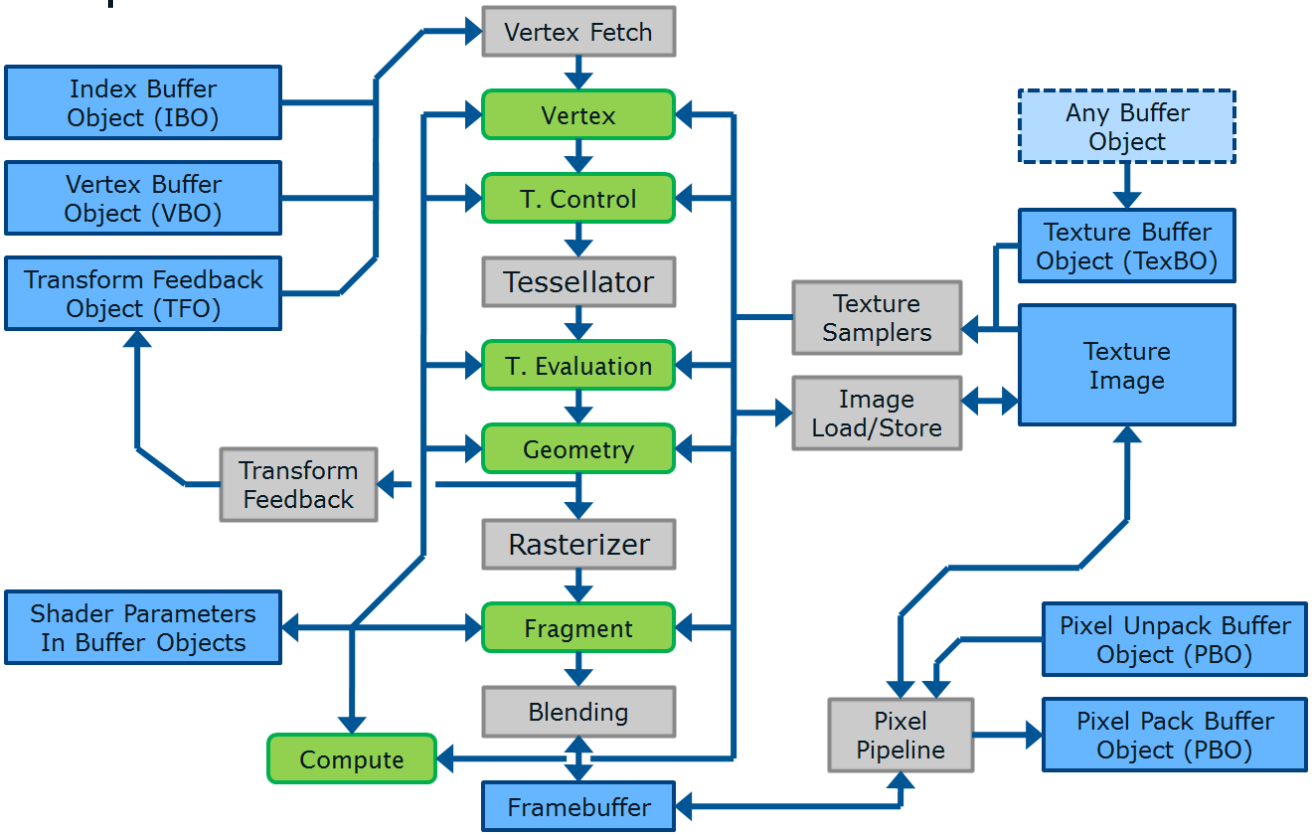


Tessellation

Tessellation

Rendering Pipeline

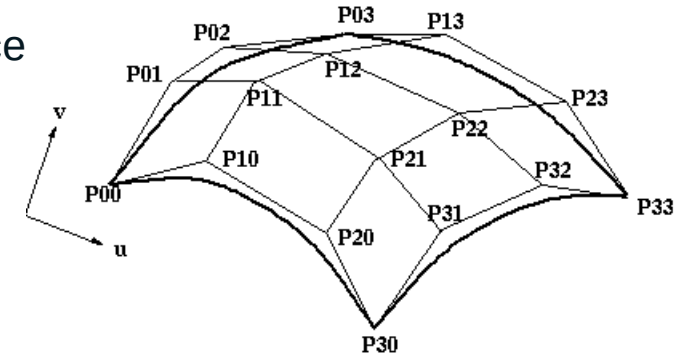
OpenGL 4.4 Pipeline:



New Generalized Graphic Primitive

Patch

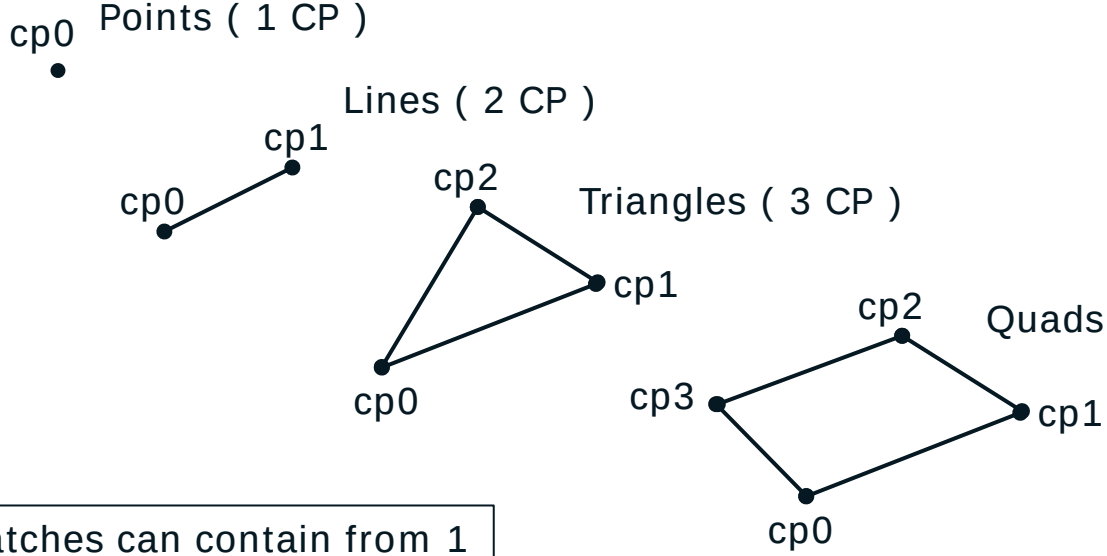
- PATCH is a generalized abstract description of surface or line
- PATCH allows to describe any simple primitive like point, line, triangle or quad
- PATCH can also describe complex surfaces like Bezier bi-cubic surface
- PATCH is constructed from Control Points
- Control Points helps describe shape of PATCH surface



- PATCHes are transparent and backward compatible
- We can treat basic input primitive as PATCH
- Then it's Vertices are treated like Control Points
- This allows to use tessellation without changes in mesh description
- In fact there is no difference between them, important is how you use them!

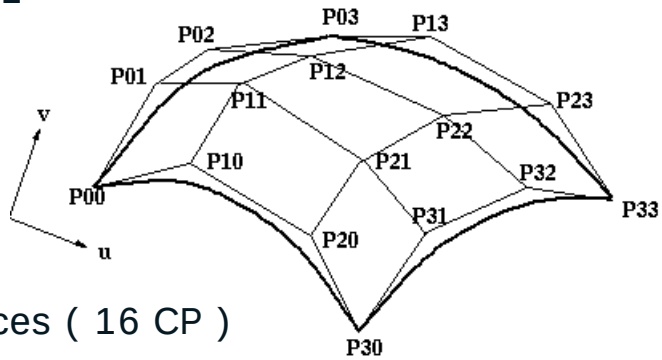
Graphic Primitives

Patch types accepted by OpenGL:



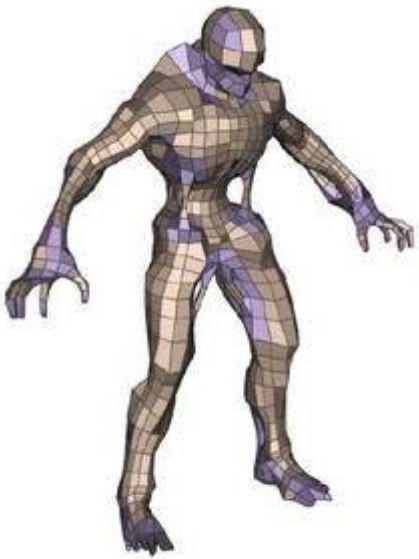
Separate Patches
(GL_PATCHES)

Patches can contain from 1 to up to 32 Control Points.



Bezier bi-cubic surfaces (16 CP)

Tessellation Shaders



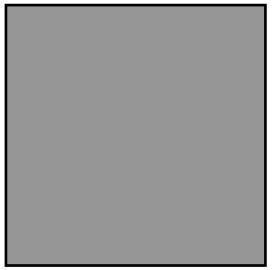
The "Imp" © Kenneth Scott, id Software 2008

Tessellation process

- Mesh is now build using Patches, in most cases Bezier bi-cubic surfaces with information about surrounding Patches, or simple Quad Patches.
- For each Patch Control Point, Tessellation Control Shader (or Hull Shader in DirectX) will calculate how much given Patch should be subdivided.
- Then Fixed Function Unit – Tessellator will generate different amount of triangles from each patch.

Tessellation Shaders

The "Imp" © Kenneth Scott, id Software 2008

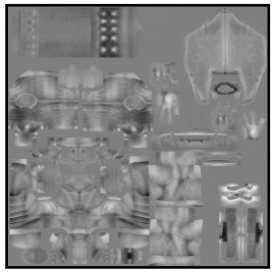


Tessellation process

- For each vertex of geometry produced by Tessellator, Tessellation Evaluation Shader (or Domain Shader in DirectX) will be executed.
- If there is no additional information available about details of geometry, TES can position new vertices in the way, triangles will be layed out in smooth gradients.
- Here displacement map is empty.

Tessellation Shaders

The "Imp" © Kenneth Scott, id Software 2008



Tessellation process

- On the right we see „Imp" from Doom 3 created by Kenneth Scott.
- In this example we have available displacement map, which can be used by Evaluation Shader to displace vertexes in given directions, creating new detail.

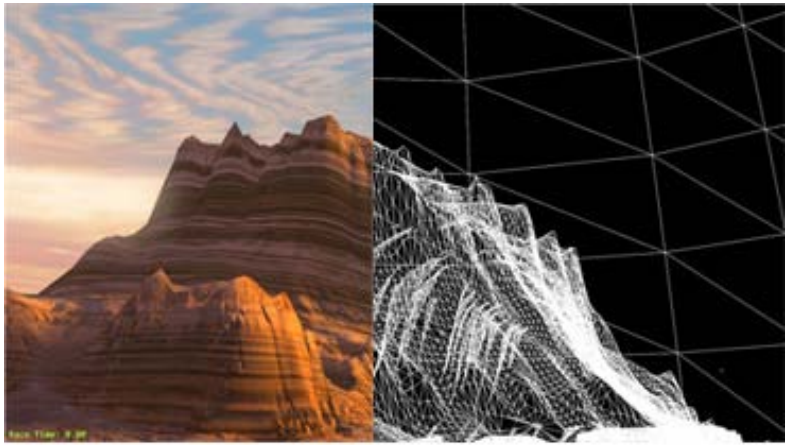
Tessellation Shaders

Deformation, transformation

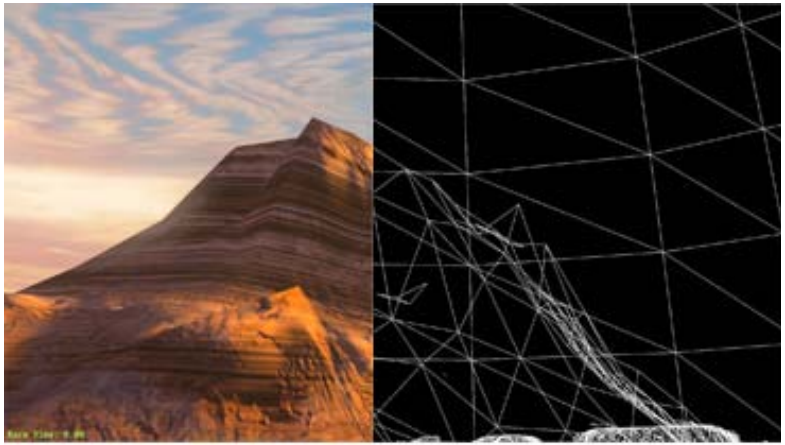
- Skin disease deformation
- Terrain modification and transformation
- Vehicles more realistic destruction

Adaptive geometry

- Amount of geometry details dynamically matched to their visibility, distance from observer
- No need for few mesh versions



Tesselation On

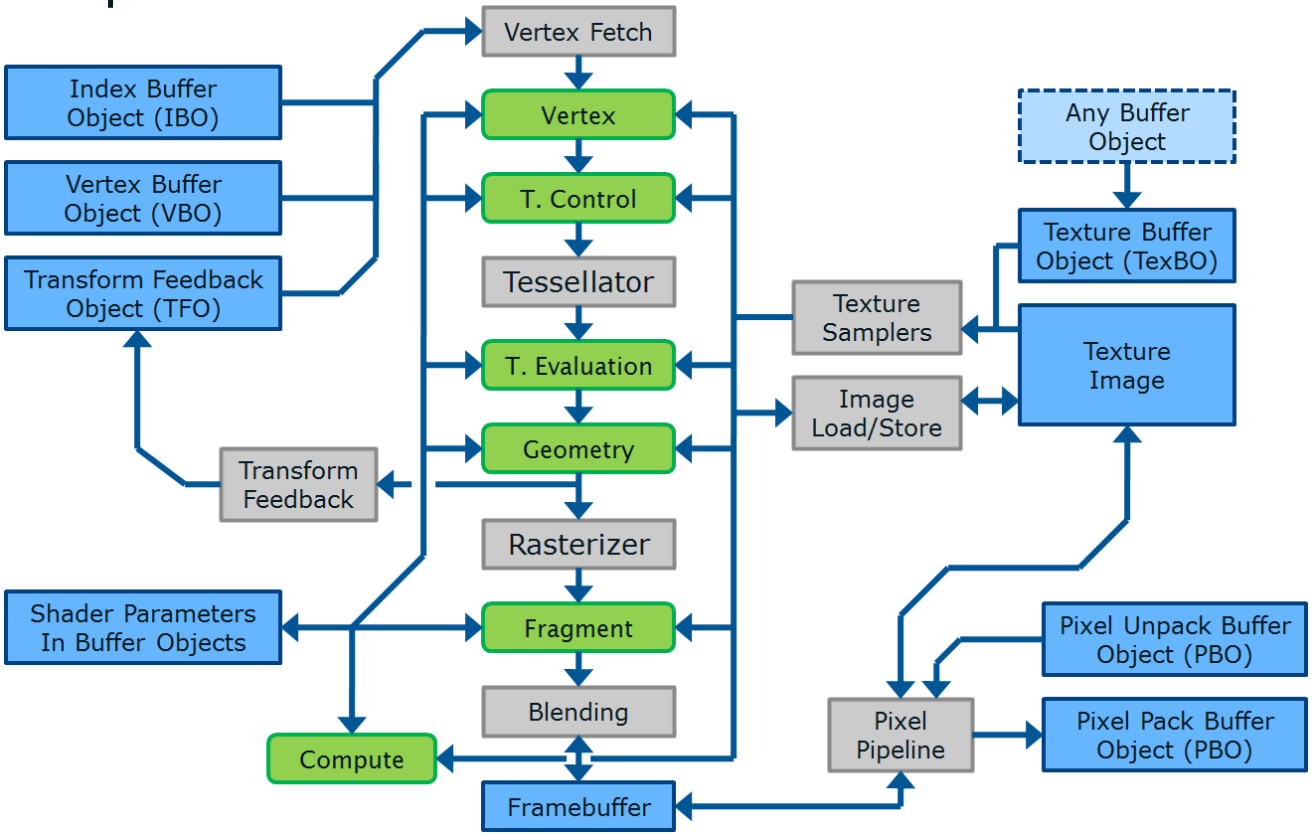


Tesselation Off

Geometry Shader

Rendering Pipeline

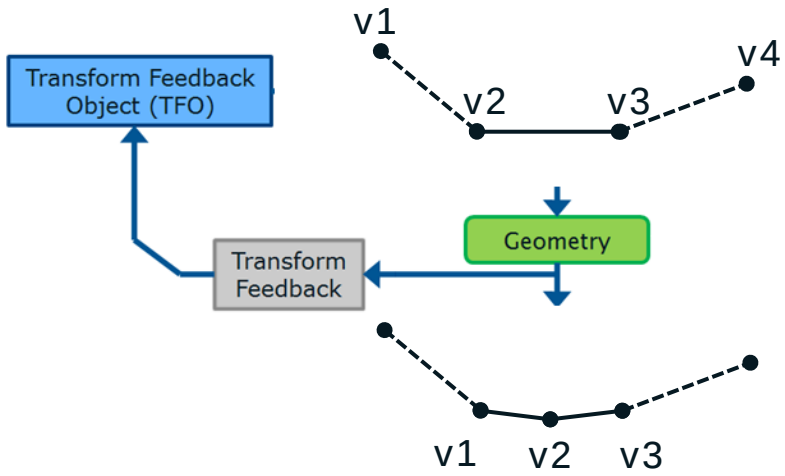
OpenGL 4.4 Pipeline:



Geometry Shader

Making lines look smoother (perfect for hair):

IN: Separate Lines with Adjacency
(GL_LINES_ADJACENCY)

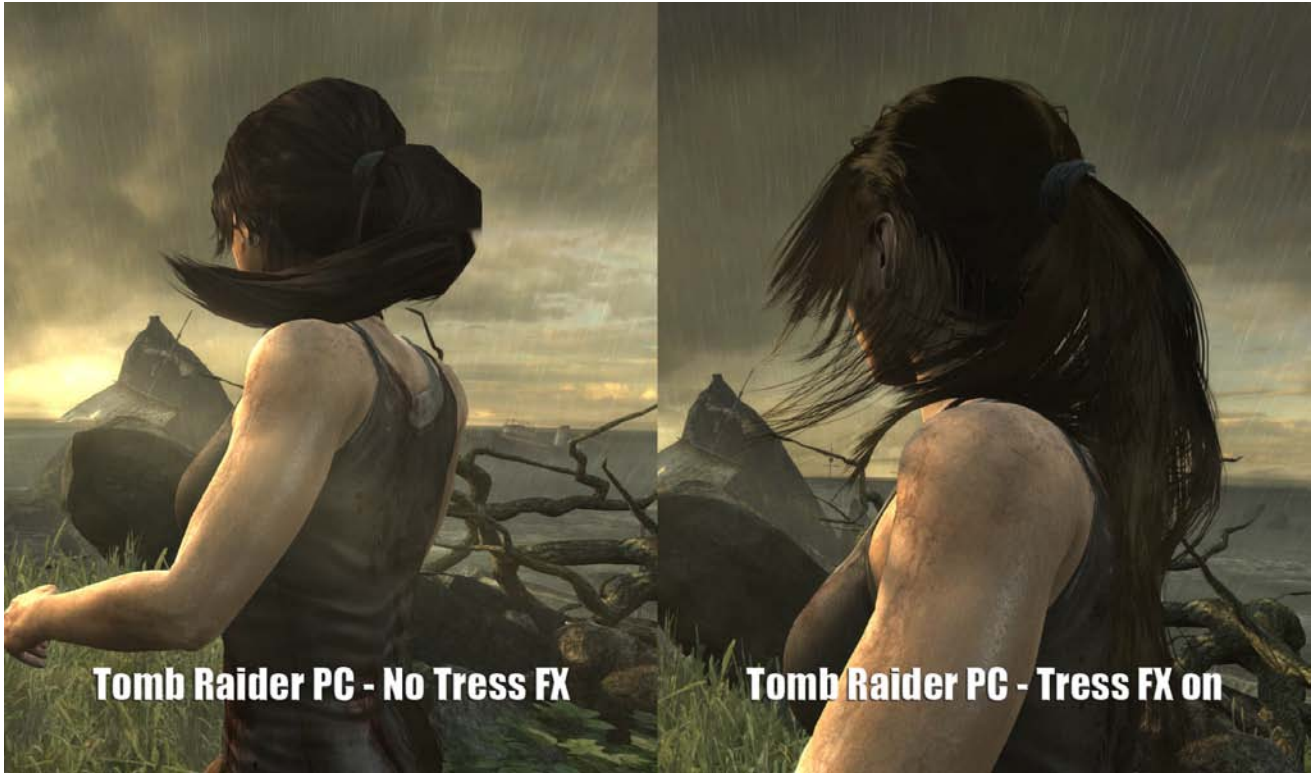


OUT: Line Stripes
(GL_LINE_STRIP)

Small set of lines is made more dense in Geometry Shader and modified to take into notice wind and motion. Resulting geometry is dense and represents hairs in current game frame.

Geometry Shader

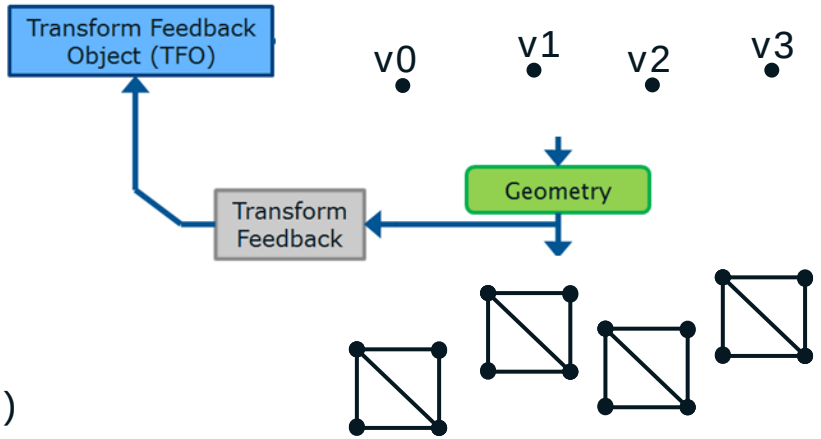
Making lines look smoother (perfect for hair):



Geometry Shader

Transforming set of points into quads (particles):

IN: Points
(GL_POINTS)



OUT: Triangle Stripes
(GL_TRIANGLE_STRIP)

Fire, water or other particles are stored as array of points. After processing their new positions, they are changed to quads on screen with mapped textures to them. Saving results through Transform Feedback allows continuous simulation.


UNREAL
ENGINE

Geometry Shader

Particle systems:

- Hair rendering
- Water, fire, smoke, dust, foam, clouds, explosion debris etc.

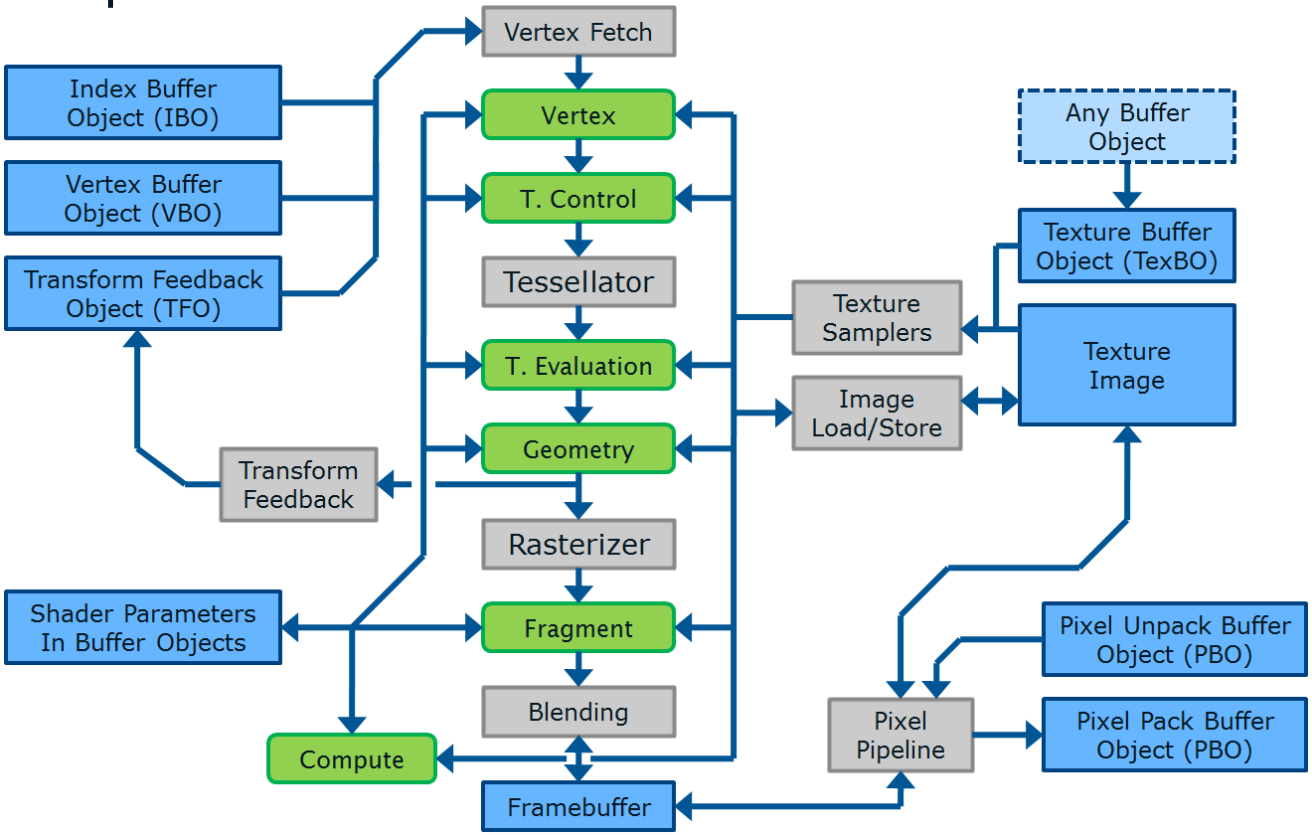
Geometry output:

- Allows procedural geometry by it's gradual update in time
- General purpose algorithms on Hardware without Compute Shaders support


Fragment Shader

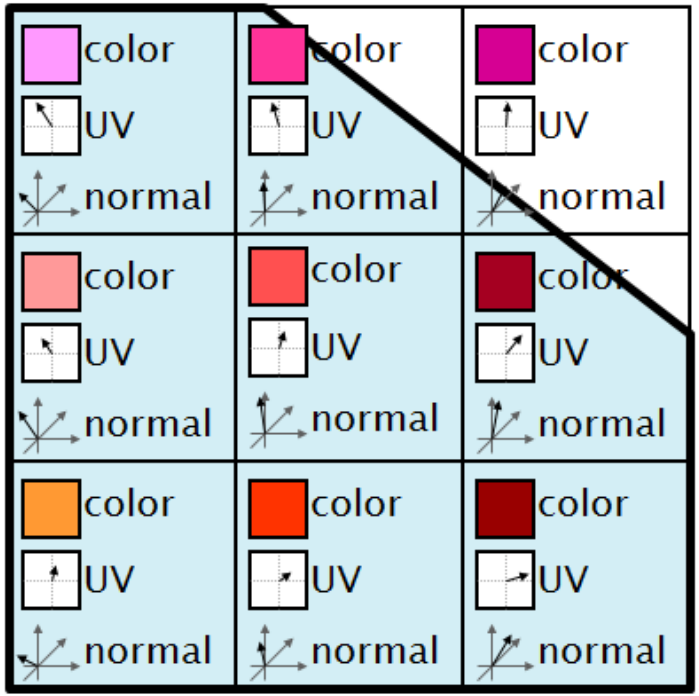
Rendering Pipeline

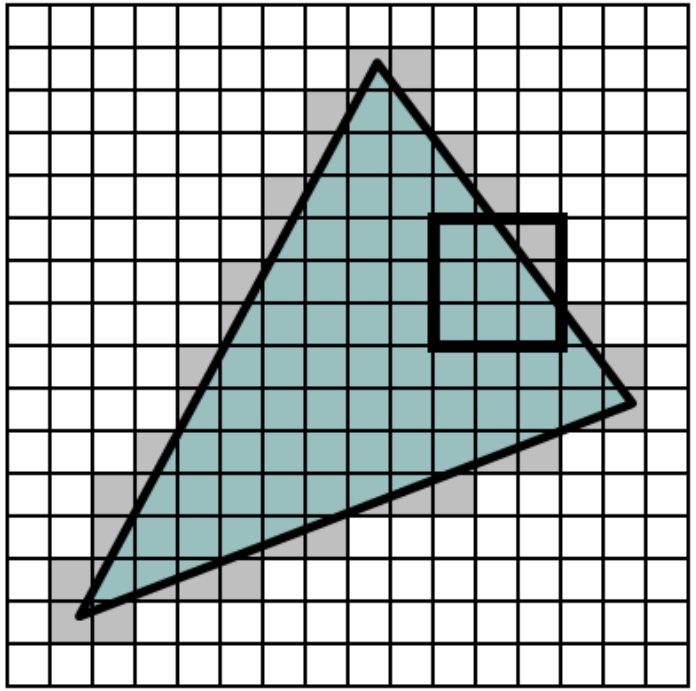
OpenGL 4.4 Pipeline:



Rasterization

Geometry is now in screen space. It is time to convert it to pixels and compute their final colors.





Fragment Shader

Geometry parameters are interpolated for each pixel.

Image source: „Deferred Rendering in Killzone 2” Guerilla Games

Fragment Shader

Geometry parameters are interpolated for each pixel.

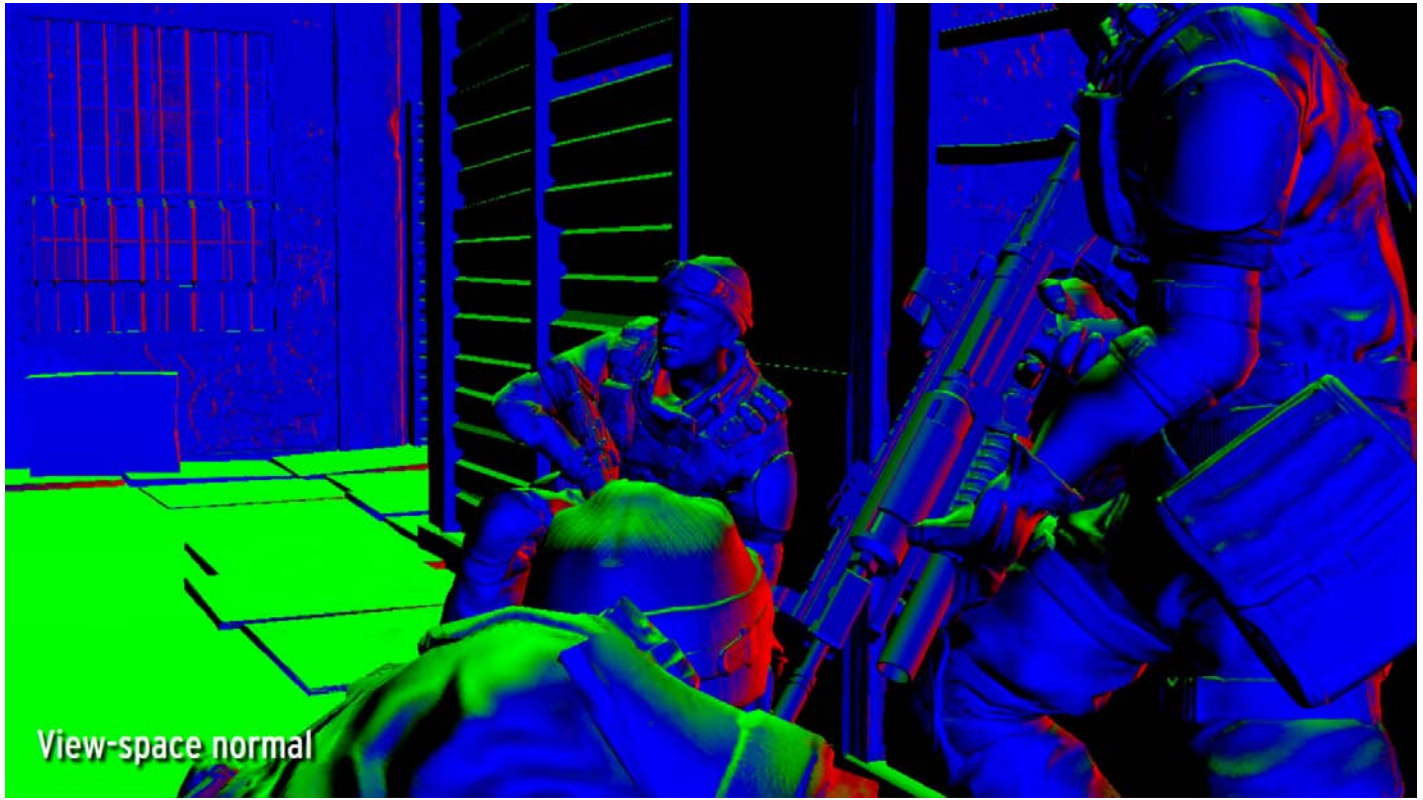


Image source: „Deferred Rendering in Killzone 2” Guerilla Games

Fragment Shader

Geometry parameters are interpolated for each pixel.

Image source: „Deferred Rendering in Killzone 2” Guerilla Games

Fragment Shader

Geometry parameters are interpolated for each pixel.

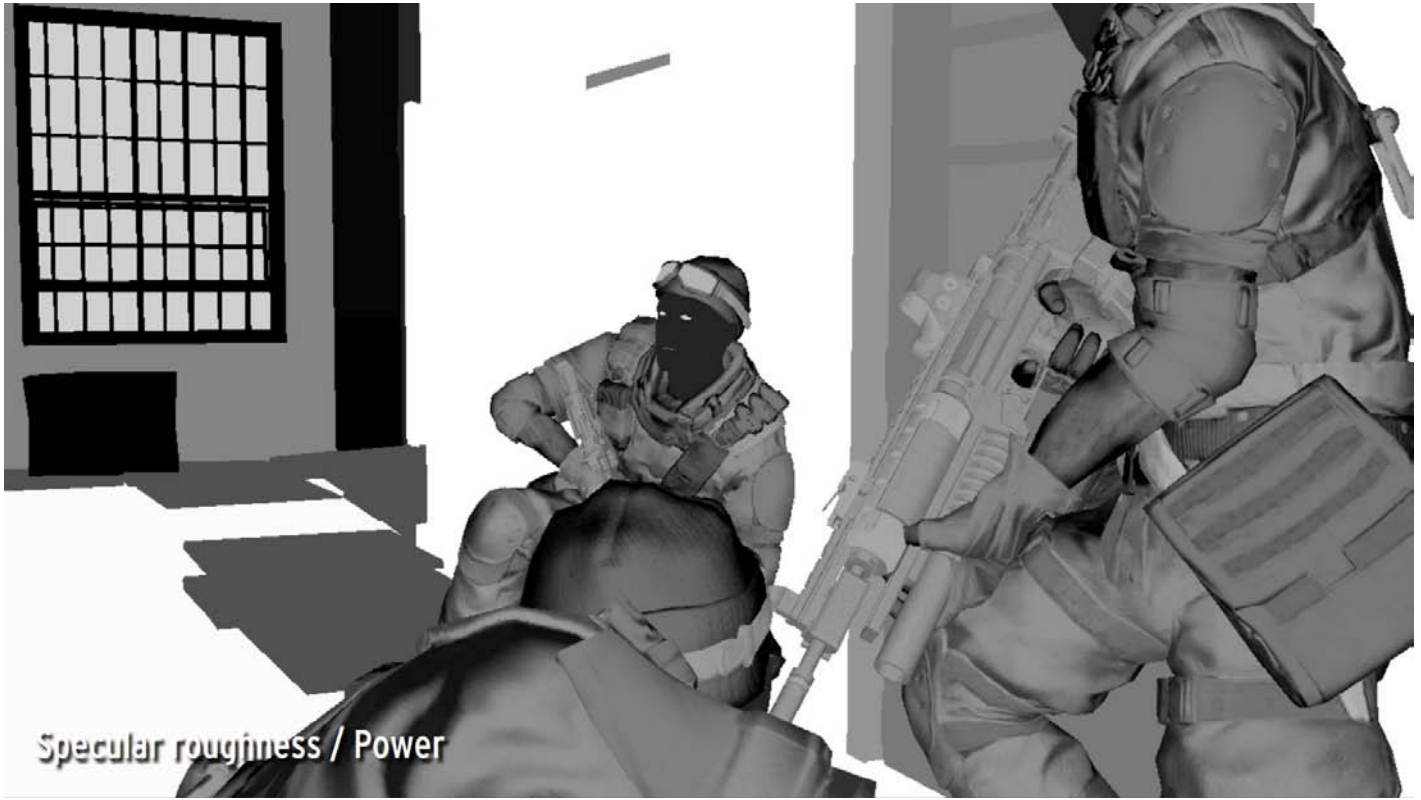


Image source: „Deferred Rendering in Killzone 2” Guerilla Games

Fragment Shader

Geometry parameters are interpolated for each pixel.

Image source: „Deferred Rendering in Killzone 2” Guerilla Games

Fragment Shader

Geometry parameters are interpolated for each pixel.

Image source: „Deferred Rendering in Killzone 2” Guerilla Games


normal maps applied
normal, spec, diffuse
maps applied
~35000 polys

Fragment Shader (Pixel Shader)

Colors Calculation

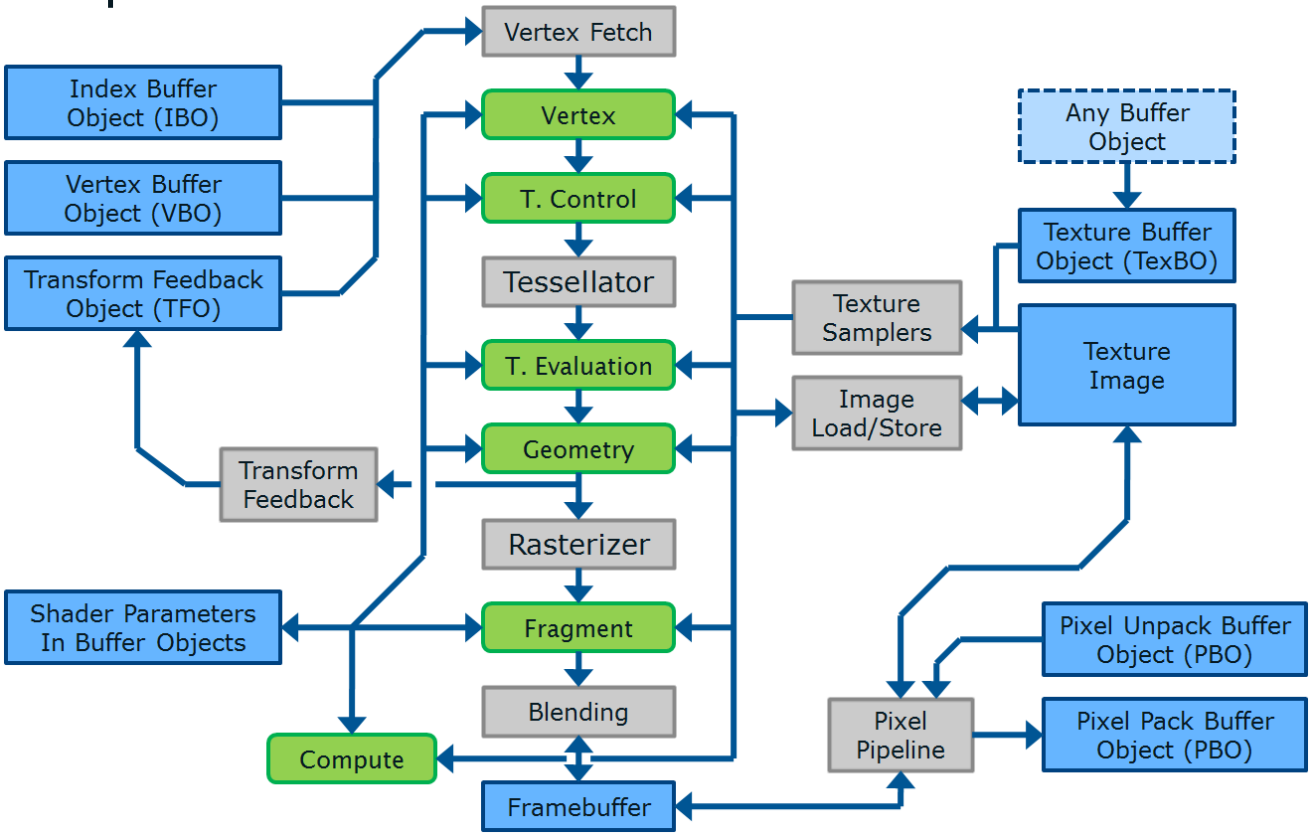
- Shading surfaces of geometry visible on screen, to enhance visual quality of rendered image.

- Simple texture mapping
- Increasing surface details
- Illumination models
- Procedural algorithms


Compute Shader

Rendering Pipeline

OpenGL 4.4 Pipeline:



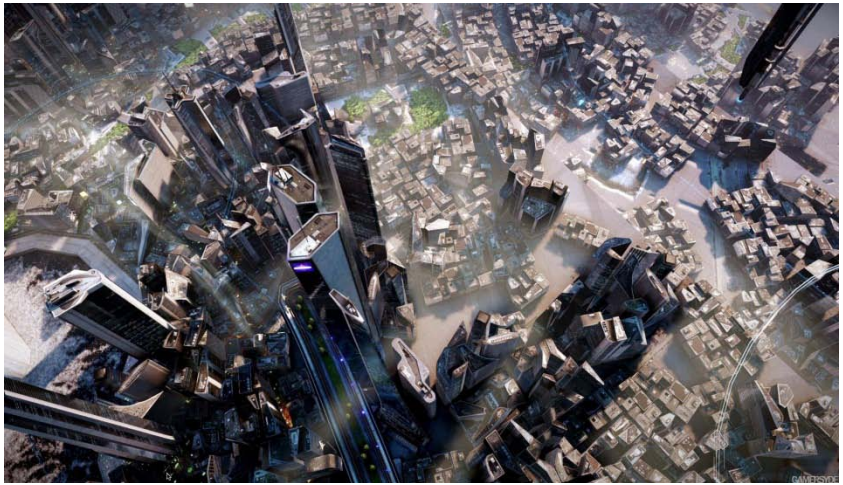

POST PROCESSING
Unprocessed image


POST PROCESSING
Cinematic color grading & depth of field

Compute Shader

Post-processing effects

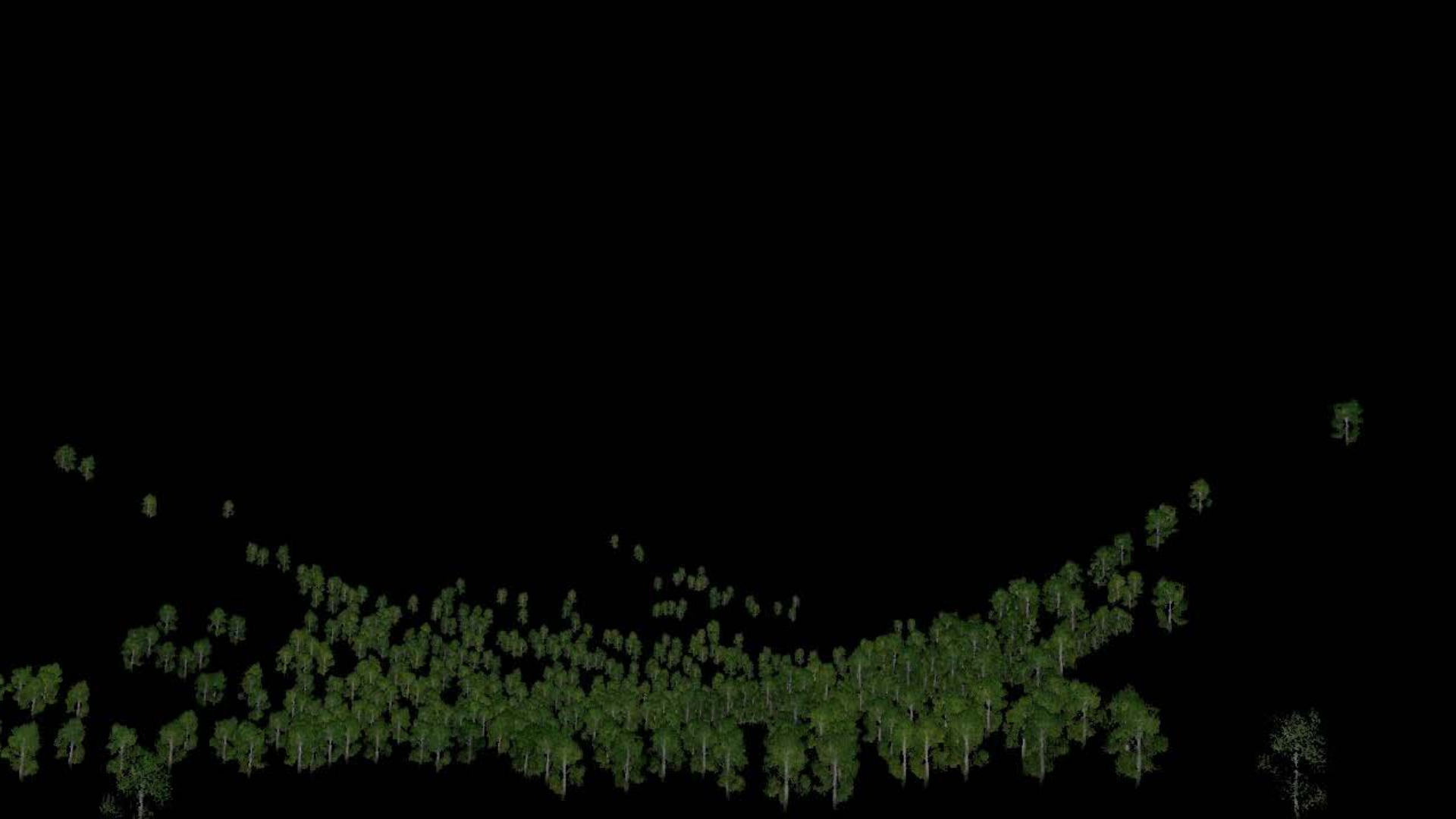
- Apply lighting techniques to enhance the mood in a scene. Secondary light bounces and Global Illumination.



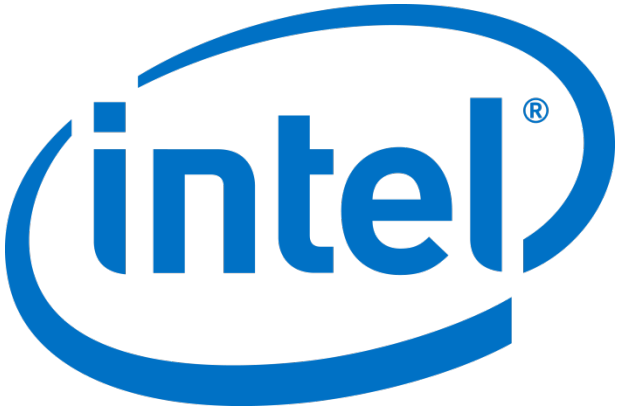
Depth of field and defocus

- More realistic transitions of focal points – imagine looking through a gun sight or a camera lens. Bokeh DOF algorithms.

Conclusion



Thank You


intel