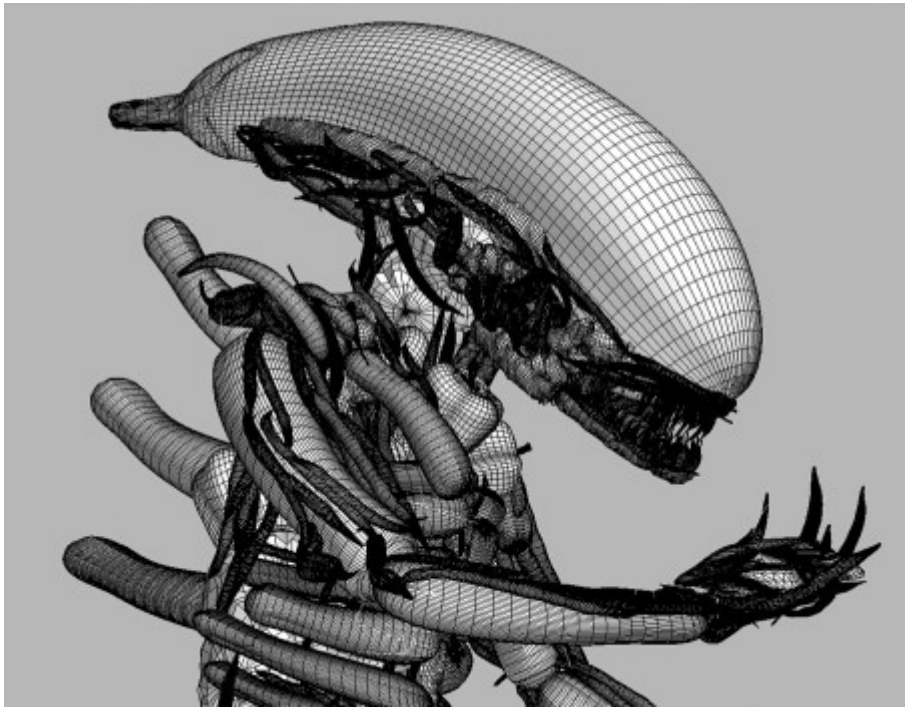
Real - Time Rendering

Pipeline optimization

Michal Červeňanský
Juraj Starinský

Motivation

- Resolution 1600x1200, at 60 fps
- Hw power not enough
- Acceleration is still necessary



Overview

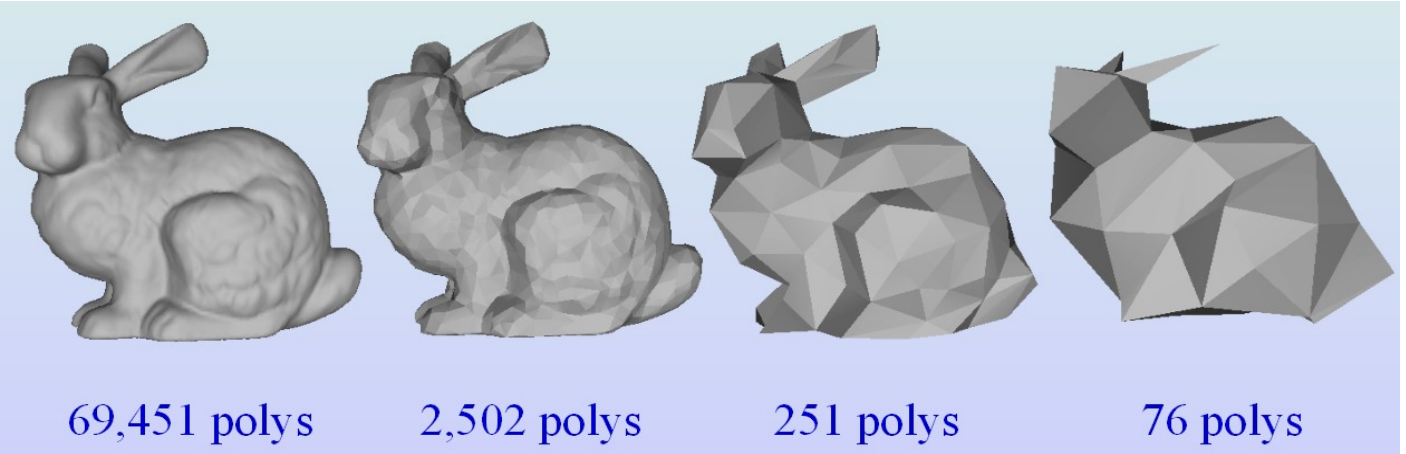
- Application stage
- Geometry stage
- Rasterization stage
- Bottleneck finding
- Tips & tricks
- Programming techniques

Application stage

- Application stage
  - LOD
  - Culling
  - Model description
- Geometry stage
- Rasterization stage
- Bottleneck finding
- Tips & tricks
- Programming techniques

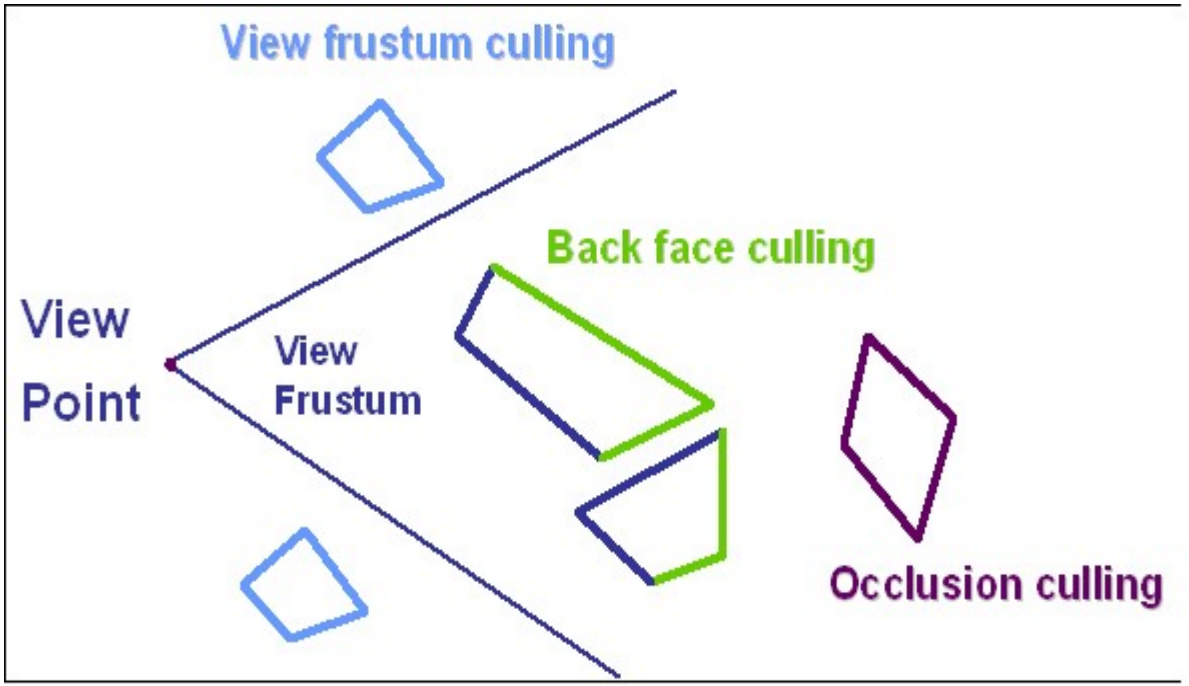
LOD

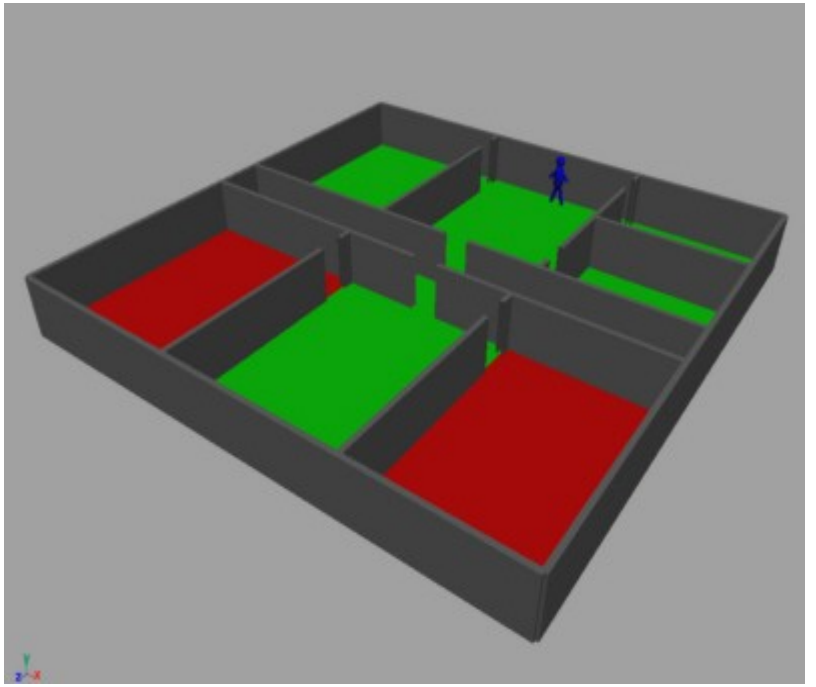
- Levels of detail
  - Simplification
  - Refinement (Tessellation)
- Models
  - Large - subdivide
  - Small - combine
- Various techniques



Culling

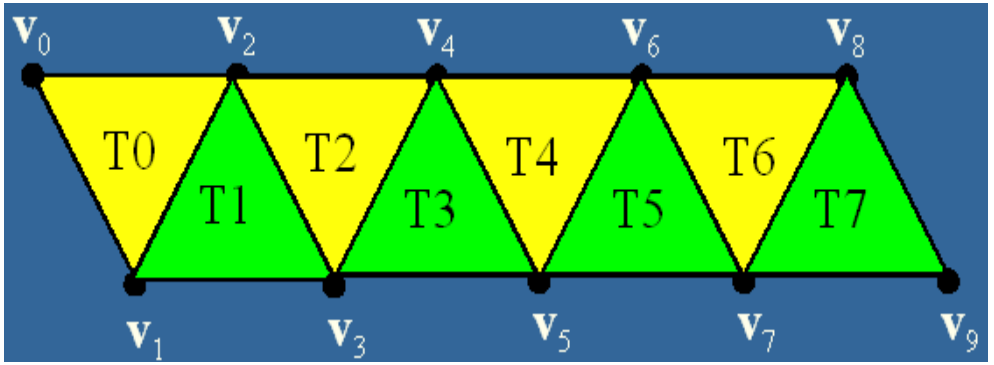
- Application stage or HW support



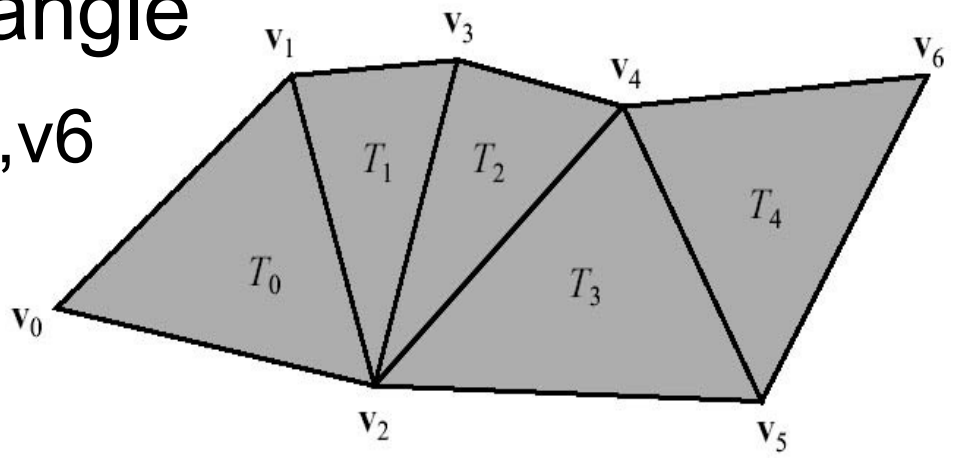


Model description

- Triangle strips
  - with: 10 vertices
  - without: 24 vertices

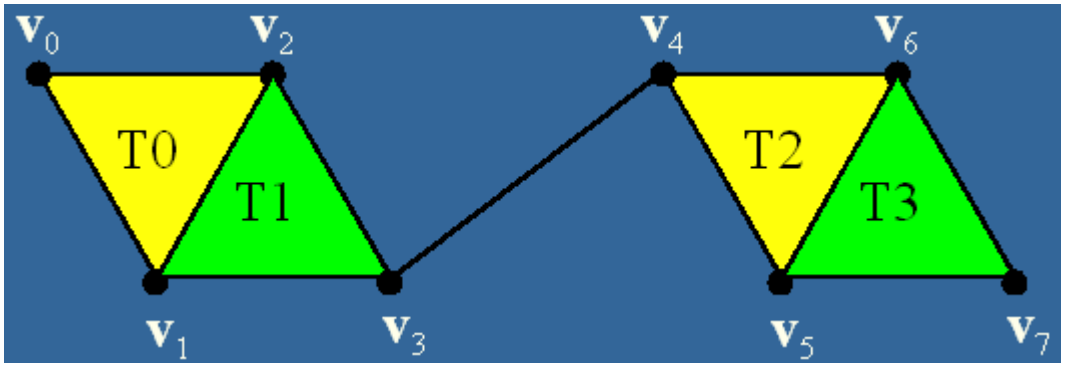


- Orientation change
- Swaps – degenerate triangle
  - v0,v1,v2,v3,v2,v4,v5,v6



Model description (2)

- Non-connected strips – swaps 0,1,2,3,3,4,4,5,6,7
- Creation
  - Manually
  - NVTriStripper
  - Own code
- Vertex arrays
  - Example



Geometry stage

- Per-vertex operations
- Triangle strip
- Lightning
  - Disable / enable
  - # light sources
  - Spot, point, directional
- Normals – normalize (preprocessing)
- Static light & diffuse material
  - Precomputed lightning

Rasterization stage

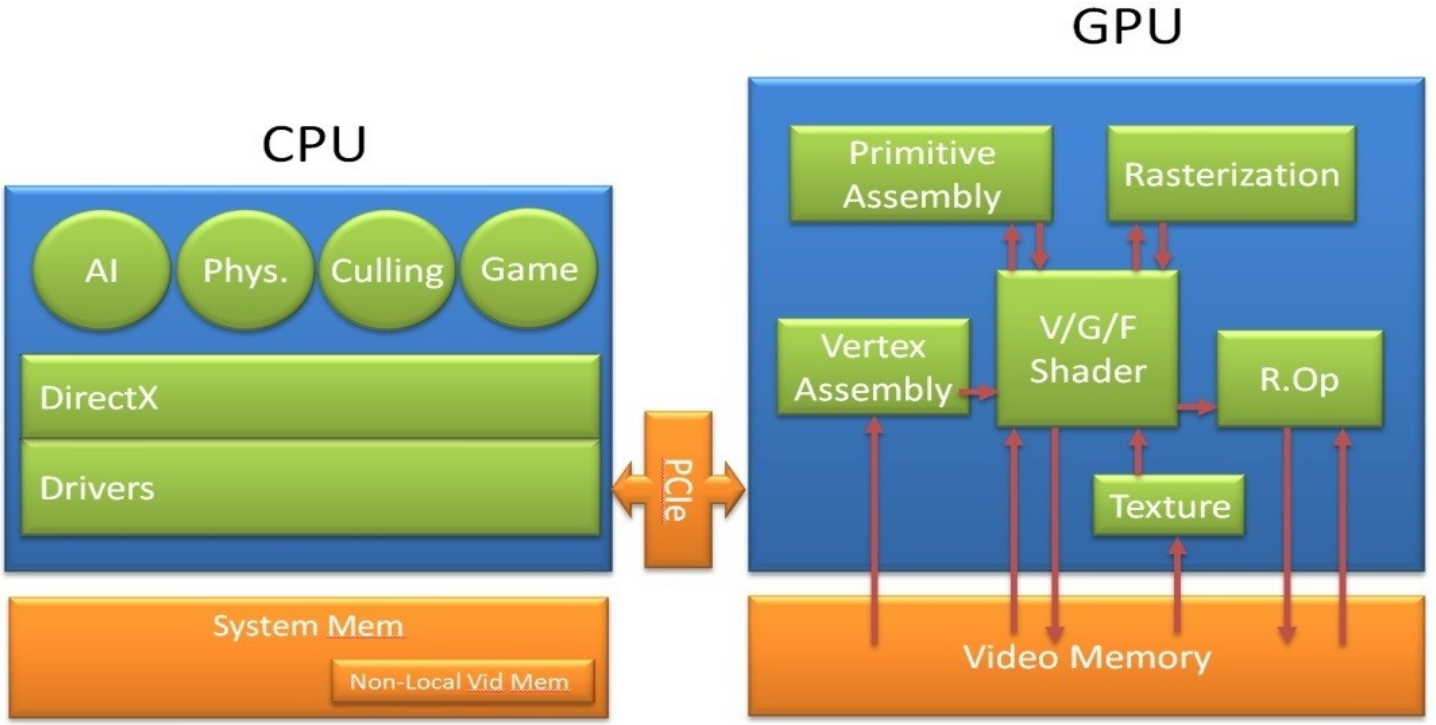
- Per-pixel operations
- Backface culling – turn on
- Draw in front to back order
- Disable unnecessary features
  - Texture filtering
  - Fog
  - Blending
  - Multisampling
- Render to texture – smaller resolution

Overall optimization

- Reduce #primitives
- Turn off features not in use
- Minimize state changes – API calls
- Color (depth) buffer clear – if necessary
- Reads to CPU – expensive
- New HW features – VBO,PBO,FBO
- Memory optimizations (order, align, …)
- Disable vertical sync
- Run on the target HW
- Good algorithm at first !

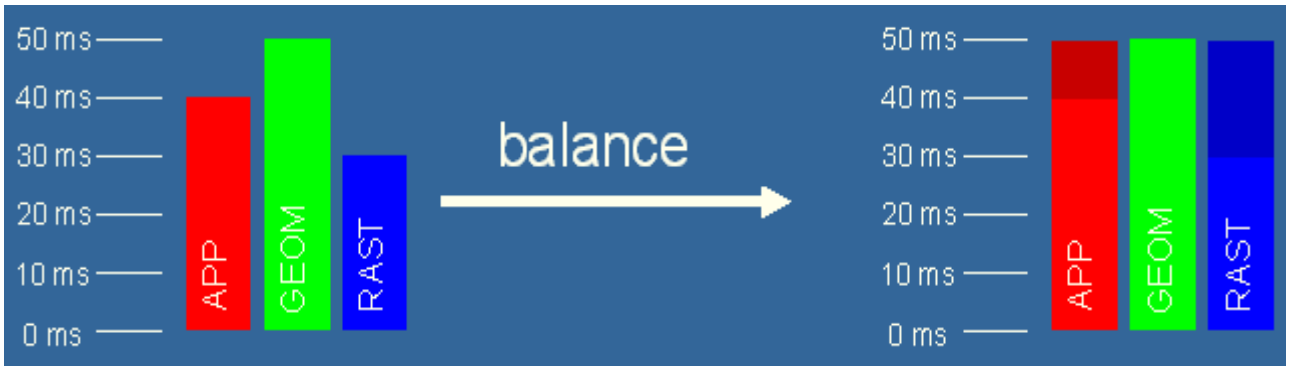
Bottleneck finding

- Rendering - as fast as slowest unit
  - Application limited (AI, collision detection,...)
  - Rasterization limited (high resolutions)
  - Geometry limited (scientific app.)



Basic tests

- Eliminate all file accesses – hdd usage
- Down-clock (BIOS, Coolbits, PerfHUD)
  - CPU speed
  - GPU speed (shaders, core)
  - GPU memory speed



CPU bottleneck

- Task manager, top
- CPU consumers (AI, physics, ...)
- Algorithmic improvements
- API calls → reduce state changes
- Locking resources → synchronization
- Protect the GPU (culling, ...)
- Increase batch size (shaders, gometry, …)
- Texture atlas, texture array, texture formats

GPU bottleneck

- Geometry
  - Changes also in rasterization
  - # lights
- Rasterization
  - Window size
  - Turn off texturing, blending, ...

Tips & tricks

- API
- Vertex processing
- Shaders
- Texturing
- Rasterization

T&T – API

- Batching
- Texture atlases, texture arrays
- State changes
- Shader constants → groups

T&T – Vertex processing

- Indexed primitives (cache)
- Recalculate data in VS (size of → vertex cache)
- Reduce vertex size
- Dynamic vertex buffers if necessary

T&T – Shaders

- Use highest profile (compiler, bug fix)
- Lowest precision → half vs. float
- Linearizable calculations → vertex shader
- Early Z culling
- Vertex operations to pixel shader
- Geometry shader if necessary
- GS → smallest maxvertexcount, smallest vertices
- Constants → hard coded
- Save computation → use algebra
- Complex functions → texture lookup
- Branching – tile size

T&T - Texturing

- Mipmaping → always
- Trilinear, anisotropic filtering → prudently
- Texture atlas
- Per-pixel lightning → precomputed in texture
- Texture format

T&T - Rasterization

- Depth, stencil rendering → double speed
  - Color writes → disabled
  - Texkill (discard, clip)
  - Depth modification
  - Alpha test
  - Occlusion query
- Early Z (Z-cull) - Front to back rendering
- Deffered shading
- Memory
  - Render targets, shaders, textures (size, priority)

Programming examples

- VBO – Vertex buffer object
  - Mainly for geometry rendering
  - Index buffer
  - Geometry processing
- PBO – Pixel buffer object
  - Buffer for transfer pixel data
- FBO – Frame buffer object
  - Render into custom frame buffer

References

- Real-time rendering slides
- GeForce 8 and 9 Series GPU Programming
  - http://developer.nvidia.com/object/gpu_programming_guide.html
- http://en.wikipedia.org/wiki/
- http://flickr.com/photos/buou/2126755136/
- http://vr.kaist.ac.kr/_r011.htm
- http://www.vis.uni-stuttgart.de/eng/teaching/lecture/ws04/seminar_spiele/bsp/
- PBO
  - http://www.mathematik.uni-dortmund.de/~goeddeke/gpgpu/tutorial3.html
  - http://www.songho.ca/opengl/gl_pbo.html

Questions ?