Realtime Computer Graphics on GPUs

Framebuffer and Offscreen Rendering Techniques

Jan Kolomazník

Department of Software and Computer Science Education
Faculty of Mathematics and Physics
Charles University in Prague

Computer Graphics Charles University

Introduction

Definitions and History

- Framebuffer, screen buffer, video buffer, . . .
- Memory containing bitmap driving video display
- 70s – framebuffers big enough to contain standard video image
- Atari 2600 – *Racing the beam*
- HW support for sprites, shifting the framebuffer (scrolling), . . .

Double Buffering

- Single frame buffer problems:
  - screen tearing
  - flickering
  - render artefacts
- Double buffering - also known as *page flipping*

  Front buffer – currently visible

  Back buffer – currently rendered off-screen
- Requires fast buffer swap

Double Buffering

- Single frame buffer problems:
  - screen tearing
  - flickering
  - render artefacts
- Double buffering - also known as *page flipping*

  Front buffer – currently visible

  Back buffer – currently rendered off-screen
- Requires fast buffer swap

Framebuffer Structure

FRAMEBUFFER

- Default framebuffer created with window creation
- Custom off-screen framebuffer:
  - Can choose resolution
  - Arbitrary attachments
  - Render to texture
  - Filtering, postprocessing
  - Interoperability with other APIs (CUDA, OpenCL, ...)

Framebuffer

- Default framebuffer created with window creation
- Custom off-screen framebuffer:
  - Can choose resolution
  - Arbitrary attachments
  - Render to texture
  - Filtering, postprocessing
  - Interoperability with other APIs (CUDA, OpenCL, ...)

Framebuffer Attachments

- 2D rendering target
- Almost any object containing image or image array
- For complex objects specify what part to attach:

  Cube map select face

  3D texture z-slice

  Mipmap choose a level

  ...
- Specify semantics – how it will be used in the rendering pipeline

Color Attachments

- Should match fragment shader outputs
- Color:
  - 1-4 channels
  - Integer (8-32), float
  - Special storage types: GL_R3_G3_B2, GL_RGB10_A2, . . .
- Color updated on successful pass through all fragment tests

Depth Buffer (Z-Buffer)

- Contains depth information for each pixel
- Solves visibility problem
  - Geometry can be streamed
  - Works only for opaque objects
- Precision depends on:
  - z-buffer element type
  - projection – decreasing precision with increasing distance (choose proper near/far clipping planes)

Depth Buffer (Z-Buffer)

- Contains depth information for each pixel
- Solves visibility problem
  - Geometry can be streamed
  - Works only for opaque objects
- Precision depends on:
  - z-buffer element type
  - projection – decreasing precision with increasing distance (choose proper near/far clipping planes)

Stencil Buffer

- Additional buffer with integer elements
- Usually shares memory with z-buffer
- Limits area for rendering – stenciling
- Often used for shadow computation
- Can be updated by results of stencil and depth test
- Behavior setup:
  glStencilFunc: what the test does
  glStencilOp: what happens on test pass/fail

Operations and Tests on Fragments

- Scissor test
- Alpha test
- Depth test
- Stencil test
- Blending
- Dithering
- Logical operations (only integer based colors)

Operations and Tests on Fragments

- Scissor test
- Alpha test
- Depth test
- Stencil test
- Blending
- Dithering
- Logical operations (only integer based colors)

Operations and Tests on Fragments

- Scissor test
- Alpha test
- Depth test
- Stencil test
- Blending
- Dithering
- Logical operations (only integer based colors)

Operations and Tests on Fragments

- Scissor test
- Alpha test
- Depth test
- Stencil test
- Blending
- Dithering
- Logical operations (only integer based colors)

Operations and Tests on Fragments

- Scissor test
- Alpha test
- Depth test
- Stencil test
- Blending
- Dithering
- Logical operations (only integer based colors)

Operations and Tests on Fragments

- Scissor test
- Alpha test
- Depth test
- Stencil test
- Blending
- Dithering
- Logical operations (only integer based colors)

Depth Test

- Different conditions for different objects (e.g. outline hidden objects)
- glDepthFunc()
  - GL_NEVER, GL_ALWAYS
  - GL_LESS, GL_EQUAL, GL_LEQUAL, ...
- Z-fighting – z-buffer precision
- glPolygonOffset()
  - Modulate z-value for specified primitives
- Early depth test optimization

ALPHA TEST

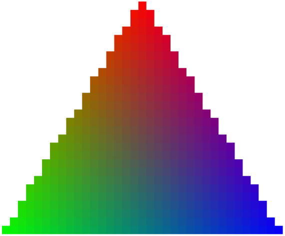
- RGBA mode – fragment accepted/rejected by the alpha test
- void glAlphaFunc(GLenum func, GLclampf ref);
- Comparison function and reference value
- By default, ref is zero, func is GL_ALWAYS
- func: GL_ALWAYS, GL_NEVER, GL_LESS, GL_EQUAL, GL_LEQUAL, GL_GEQUAL, GL_GREATER or GL_NOTEQUAL
- glEnable(GL_ALPHA_TEST);

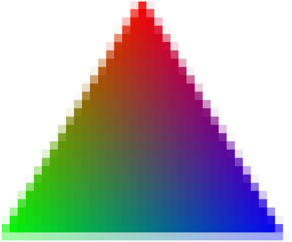
Color Blending

- How the color of the pixel is updated by fragment shader output
- Render transparent objects –
  - disable depth test, painters algorithm (order primitives)
  - order independent transparency – depth peeling
- glBlendFunc() mixing colors based on their respective alpha values.
- The source color: the color of the fragment be drawn.
- The destination color: the color already present in the color buffer.

Antialiasing

- Supersampling (SSAA)
  - Render in higher resolution
  - Show downsampled image – smoothing
- Multisampling (MSAA)
  - Multiple depth/stencil tests per pixel
  - Estimates fragment coverage – smoothing on edges





Render Buffer vs. Texture

Best buffer for framebuffer attachments?

- Render buffer object:
  - contains image, which will not be sampled (read)
  - optimized as render target
  - support MSAA
- Textures:
  - optimized for read access
  - can be used later in the rendering pipeline

Triple Buffering and V-sync

- V-Sync: new frame is rendered in sync with monitor refresh frequency (60-100 Hz)
- Double buffering + V-Sync – small interval when none of the buffers can be touched – delay, idle GPU
- Second backbuffer – no delays, highest possible framerate
- Meaningful only when refresh rate lower than maximal possible rendering framerate

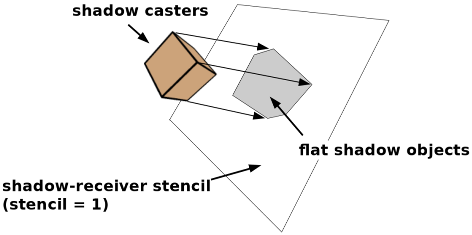
Shadows

Shadow Casting

- Static Shadows: baked light/shadow map
- Dynamic shadows:
  - single shadow-receiving plane
    - simple approach, not generally usable
  - shadow mapping
    - shadow depth-buffer, supported in HW – shadowmap sampler
  - shadow volumes
    - precise but very computationally intensive
- sharp shadows (one pass)
- soft shadows (more passes, accumulation of results)

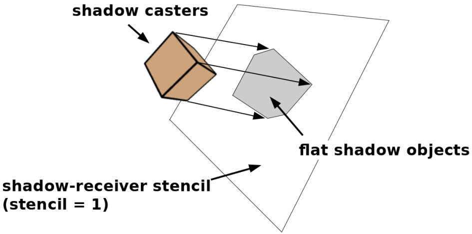
Shadow Receiving Plane

- sharp shadows – point light source
- use of stencil buffer and multiple scene passes
  - stencil prevents shadow duplication
- single shadow-receiving plane
- shadow could be opaque (destroying the original surface color) or transparent (only reducing the amount of light)



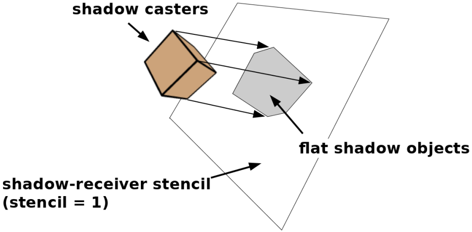
Shadow Receiving Plane – Procedure

1. the whole scene rendered using ordinary projection
   - shadow-receiver sets stencil to 1
   - other objects zero this bit
2. potential shadow-casters rendered to the shadow-receiving plane
   - depth-test is off
   - special projection matrix
   - shadows are drawn only to the (stencil==1) pixels



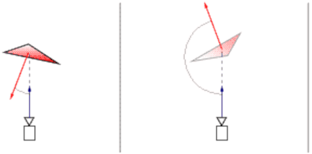
Shadow Receiving Plane – Procedure

1. the whole scene rendered using ordinary projection
   - shadow-receiver sets stencil to 1
   - other objects zero this bit
2. potential shadow-casters rendered to the shadow-receiving plane
   - depth-test is off
   - special projection matrix
   - shadows are drawn only to the (stencil==1) pixels



Face Culling

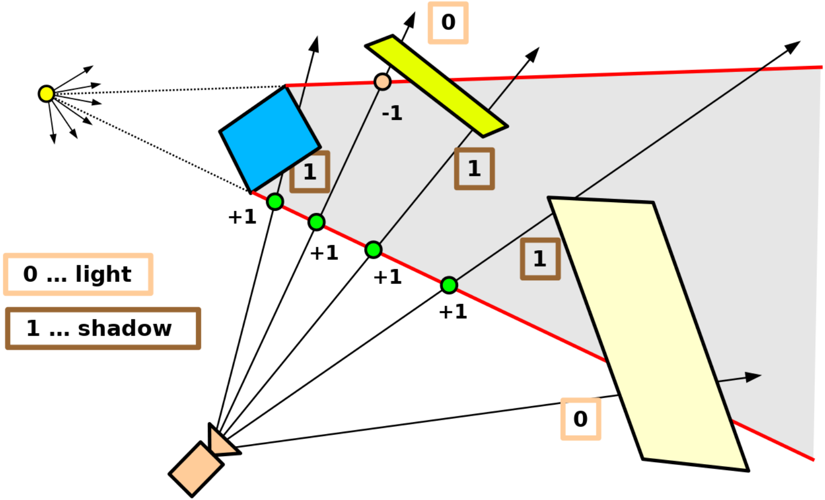
- From the point of view of camera
- GPU can filter (face cull) according to vertex order:
  - glEnable(GL_CULL_FACE);
  - glFrontFace(GL_CCW);
  - glCullFace(GL_BACK); // draw front faces only
- Speed optimization



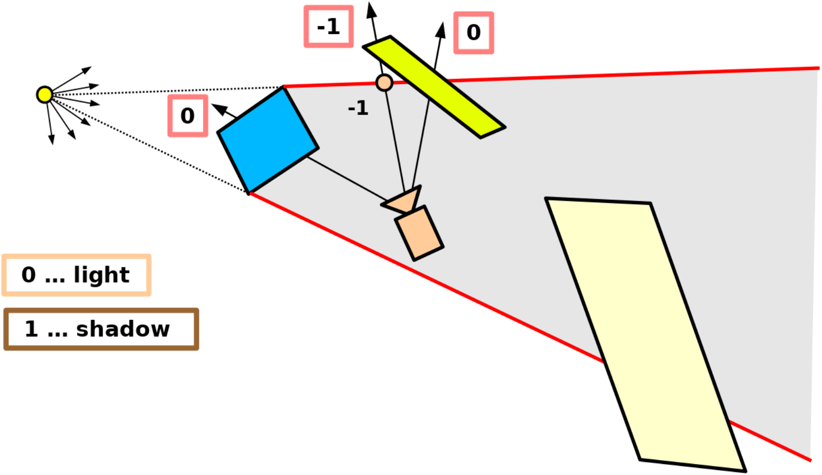
Shadow Volume – Depth Pass

- shadow-caster – infinite shadow volume from countour (shadow solid)
- lateral faces of a shadow solid are considered, but invisible
- virtual ray from the camera is tested against these faces
- GPU can rasterize the virtual faces and ”draw” them into the stencil buffer
  - Front faces increase stencil
  - Back faces decrease stencil
- stencil buffer values define shadows in the scene
- has to be done separately for each point light source

Shadow Volume – Depth Pass



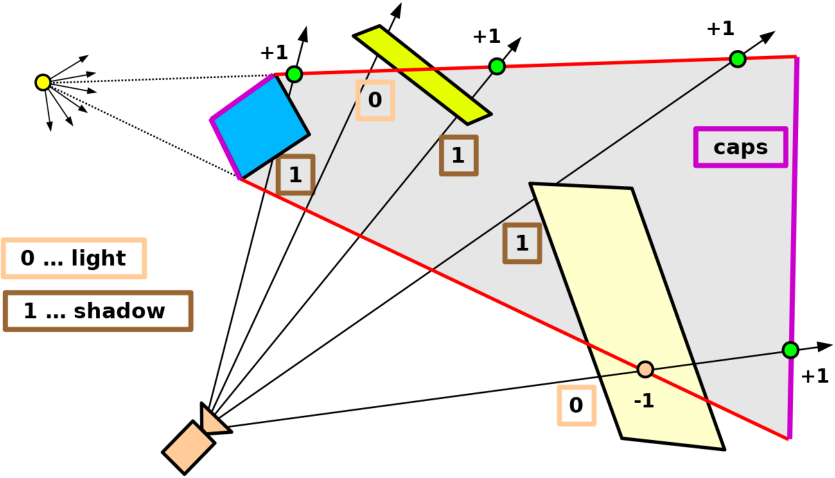
Shadow Volume – Depth Pass



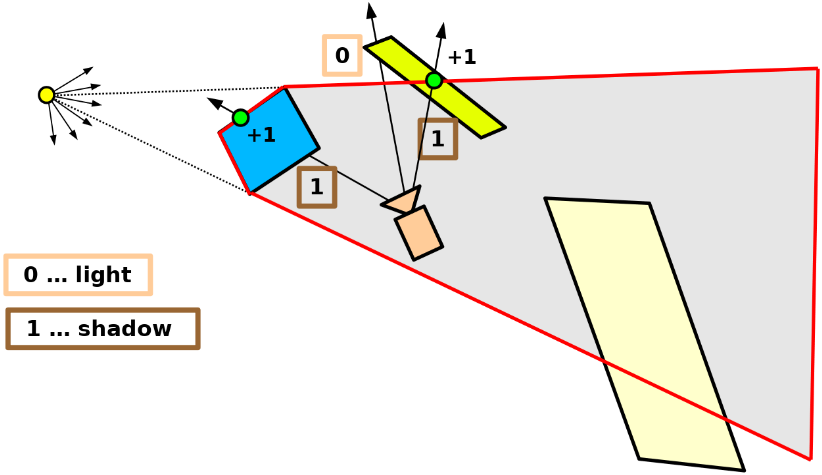
Shadow Volume – Depth Fail

- Carmack's reverse
- camera can be placed anywhere
- shadow solid sealed using ”caps”: one is illuminated part of an object, the second one in infinity
- second phase: lateral shadow faces and both ”caps”
  - Front faces – decrement on depth fail
  - Back faces – increment on depth fail
- third phase: stencil==0 means ”light”

Shadow Volume – Depth Fail



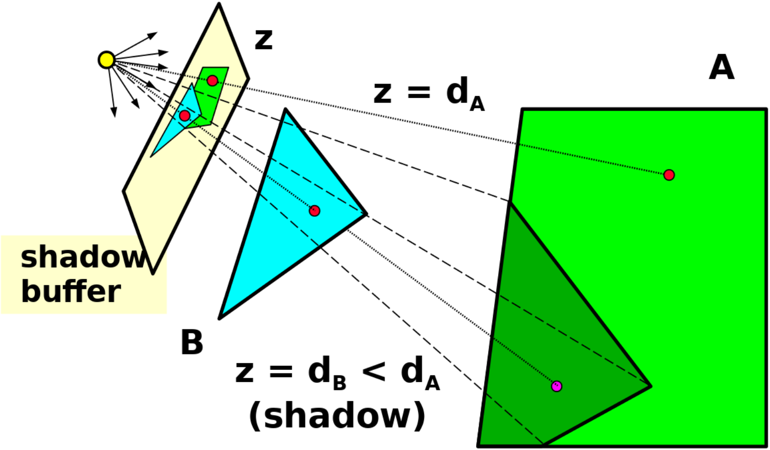
Shadow Volume – Depth Fail



Shadow Mapping

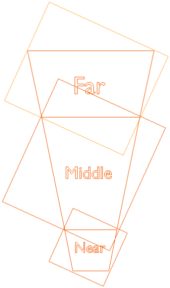
1. scene is rendered from the light-source viewpoint
   - no need to modify frame buffer, only depth-buffer has to be updated
2. depth-buffer is moved into a texture (”shadow map”)
   - regular projection according to the camera
   - use of projective texture coordinates
   - test actual distance of a fragment from the light source (in the world space) against shadow-map texture

Shadow Mapping



Shadow Mapping Problems

- Shadow acne
- Perspective aliasing
- Sharp shadows
- Hard to choose optimal size of shadow maps
  - Solution: cascaded shadow maps



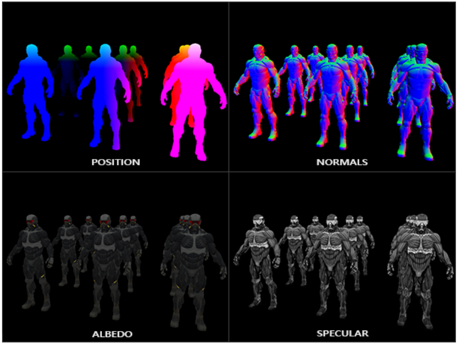
Deffered Shading

Bottlenecks in Rasterization Pipeline

- Processing lots of lights
- Complicated materials
- Lots of fragments shaded and not used

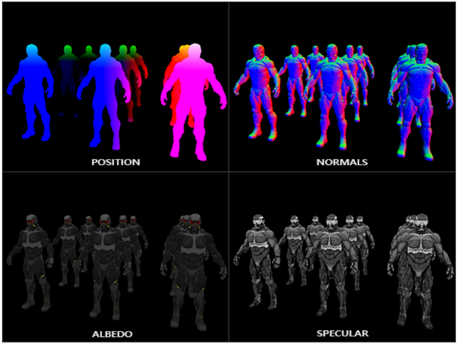
Deffered Shading

- Decouple geometry and light processing
- Two stages:
  1. Render geometry to textures – multiple render targets (G-buffer)
  2. Posprocessing – apply light computations



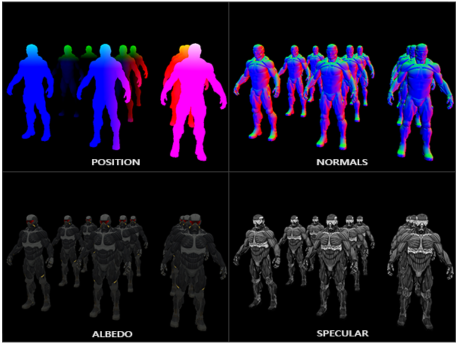
Deffered Shading

- Decouple geometry and light processing
- Two stages:
  1. Render geometry to textures – multiple render targets (G-buffer)
  2. Posprocessing – apply light computations



Deffered Shading

- Decouple geometry and light processing
- Two stages:
  1. Render geometry to textures – multiple render targets (G-buffer)
  2. Posprocessing – apply light computations



Compositing Step

- Compute shader or draw one fullscreen quad
- Apply lighting for only visible fragments
- All shading parameters come from uniforms and textures
- Modern engines do postprocessing
  - Motion blur
  - Depth of field
  - Screen space ambient occlusion
  - Screen space decals
  - Bloom
  - HDR processing

Disadvantages

- Cannot handle transparency (depth peeling)
- Complicated usage of multiple material types
- Memory intensive
- MSAA does not work:
  - Supersampling
  - Smoothing trick (small scale, rotate with linear interpolation, ...)
  - Postprocessing – edge detection and masked smoothing, morphological AA (MLAA)

Summary: OpenGL Calls

Framebuffer setup: glGenFramebuffers, glBindFramebuffer, glGenRenderbuffers, glFramebufferTexture*, glBlitFramebuffer, glRenderbufferStorageMultisample

Z-buffer and stencil buffer: glDepthFunc, glStencilMask, glStencilFunc, glStencilOp, glPolygonOffset

Other: glBlendEquation, glBlendFunc, glScissor,