Realtime Computer Graphics on GPUs

Effects II

Jan Kolomazník

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

Advanced Texturing

Bottlenecks of Modern Renderers

- Memory transfers between CPU (RAM) and GPU
- Communication with driver:
  - Fixed pipeline:
    - Lots of API calls to manage state
  - OpenGL 3.0+:
    - Bind operations
    - Setting shader uniforms
    - Draw calls

PROBLEM – MULTI-MATERIAL SCENE/OBJECTS

- Changing shader programs + repeated uniform setup
- Bind new textures on material switch
- Multiple draw calls

Uniform Buffer Objects I

Advantages:

- Same uniforms in multiple shader programs:

```
uniform vec4 camera_position;
uniform vec4 light_position;
uniform vec4 light_diffuse;
```

- Single buffer cointaining the data
- Larger uniform storage
- Faster switching for uniform blocks

Uniform Buffer Objects II

- Switch to uniform block in GLSL

```
uniform shader_data
{
  vec4 camera_position;
  vec4 light_position;
  vec4 light_diffuse;
};
```

- C++ counterpart:

```
struct shader_data_t
{
  float camera_position[4];
  float light_position[4];
  float light_diffuse[4];
} shader_data;
```

Uniform Buffer Objects III

- Create uniform buffer:

```
GLuint ubo = 0;
glGenBuffers(1, &ubo);
glBindBuffer(GL_UNIFORM_BUFFER, ubo);
glBufferData(GL_UNIFORM_BUFFER, sizeof(shader_data), &shader_data, ↩
      GL_DYNAMIC_DRAW);
glBindBuffer(GL_UNIFORM_BUFFER, 0);
```

- Update data:

```
glBindBuffer(GL_UNIFORM_BUFFER, gbo);
GLvoid* p = glMapBuffer(GL_UNIFORM_BUFFER, GL_WRITE_ONLY);
memcpy(p, &shader_data, sizeof(shader_data))
glUnmapBuffer(GL_UNIFORM_BUFFER);
```

- Connect UBO and GLSL program:

```
block_index = glGetUniformBlockIndex(program, "shader_data");
GLuint binding_point_index = 2;
glUniformBlockBinding(program, block_index, binding_point_index);

...

glBindBufferRange(GL_UNIFORM_BUFFER, binding_point_index,
    gbo, 0, sizeof(shader_data_t));
```

Bindless Textures

How to prevent texture binding?

- Generate integer *handle* for each texture:
  - from texture object alone
  - from texture object and sampler
  - from specific image within texture
- Texture state becomes immutable (can update contents)
- Access texture by handle from shaders
  - cannot be used until made *resident*
- Safety: errors may crash the GPU, program, OS
- Extensions: ARB_bindless_texture, NV_bindless_texture

Bindless Textures – Usage

- Creation:

```
glGetTextureHandleARB(GLuint texture);
glGetTextureSamplerHandleARB(GLuint texture, GLuint sampler);
```

- Image handle:

```
glGetImageHandleARB(GLuint texture, GLint level,
GLboolean layered, GLint layer, GLenum format);
```

- Residency:

```
glMakeTextureHandleResidentARB(GLuint64 handle);
glMakeImageHandleResidentARB(uint64 handle, enum access);
glMakeTextureHandleNonResidentARB(GLuint64 handle);
glMakeImageHandleNonResidentARB(uint64 handle);
```

Bindless Textures – GLSL Usage

- Handle must be resident
- Direct use:
  - Shader stage inputs/outputs (except FS outputs)
  - Vertex attributes (GL_UNSIGNED_INT64_ARB data type)
  - Uniforms, uniform blocks

```
layout(bindless_sampler) uniform sampler2D bindless;

uniform samplers
{
  sampler2D arr[10];
};
```

  - Local sampler variables (init form other samplers, integer cast)

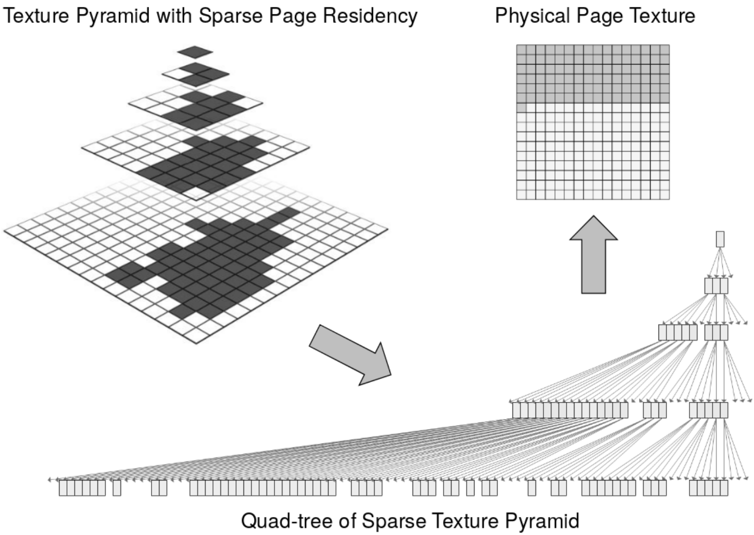
Sparse Virtual Texture

- Also known as *megatextures* (Idsoft – Rage)
- Different approach to binding prevention – one large texture for whole scene
- Texture may be larger than GPU memory (over-subscription)
  - Similar to virtual address space and physical memory
  - Pages are texture tiles
  - Page table for translation of texture coordinates
- Each object in scene uniquely textured
  - Artist less limited by technical aspects

Virtual Texturing

Virtual Texturing

Virtual Texturing



Megatextures – Page Mapping

- Access the page table with original texture coordinates (nearest neighbor)
- No special coordinate mapping
- Within-page offset:
  - Depends on mip-map level

```
page_phys_tc = texture(page_tex, vtex_tc);

within_page_tc = exp2(mip_level) * vtex_tc;
within_page_tc = fract(within_page_tc);

within_page_tc *= rescale_page_to_physical;
phys_tc = page_phys_tc + within_page_tc;

sample = texture(diffuse_tex, phys_tc);
```

Feedback Analysis

- Separate pass – render page IDs (low resolution)
  - Determine pages + mip-map levels
- Loading missing pages – delay before used (mip-map fallback)

Implementation Details

Page faults:

- Mip-map substitution
- Propagate lower mip-map levels page mapping to un-mapped upper levels

HW support:

- TexPageCommitmentARB()

Filtering:

- Bilinear filtering with/without tile borders
- Trilinear – mip-map the physical pages (larger border)
- Anisotropic – complicated

Decals, Billboards

Decals

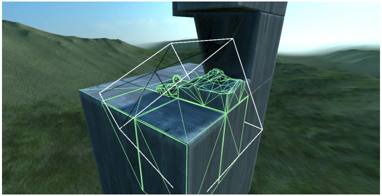
- Runtime interaction with the scene
- Additional details:
  - Bullet holes
  - Graffiti
  - Local material weathering
  - Footsteps

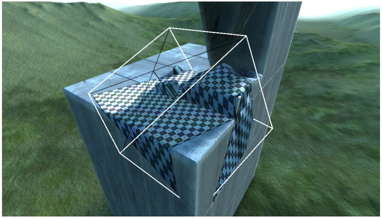
Decals – Approaches

- Megatextures:
  - Draw decals directly in the scene texture
  - Maybe permanent without increased overhead
- Special geometry rendered in front of the object
  - Z-fighting, depth offset
  - Simple scene – textured quad
  - Geometry projection in general case
  - Adding decals increases scene complexity – only few latest/important kept
- Screen space decals – deferred shading

Decals – Projecting Geometry

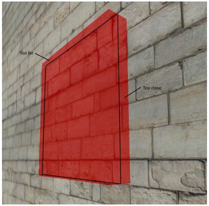
- Oriented bounding box:
  - projector along z-axis
  - x,y are mapped to u,v coordinates
- Intersection with scene geometry
  - select intersecting tringles
  - cut triangles – project to projector space, uv-mapping





Decals – Screen Space

- Deffered shading
- Render projector box
  - Reject fragments which project outside the box (use z-buffer + view direction)
  - Flattened box – projected on the geometry
- Normal mapping:
  - Normal buffer may contain modulated normals
  - Underlying geometry normal – partial derivatives in the z-buffer
- Problems:
  - Clipping the projector box
  - Projection on 90 degree corners



Billboards

- Billboard – semitransparent texture showing more complicated object/scenery
  - texture is usually mapped on a rectangle
  - often perpendicular to view direction
  - .. following the viewer – special transform matrix
  - rotation around vertical axis only (unsightly from above)
- usage
  - trees and bushes (even unoriented billboards, multi-billboards)
  - complex inscriptions, 2D graphics, HUD, lens flare..

IMPOSTORS

- Impostor – billboard created dynamically (as necessary) in a rendering engine
  - cache of complex scenery (not very dynamic)
  - complex object/scenery (geometric or color complexity)
  - for distant objects mostly
  - hierarchy, LoD, multiple instances of the (almost) same object

    ...
- trees, bushes
  - impostors might be oriented along main branches..
- technique: HW render-target textures

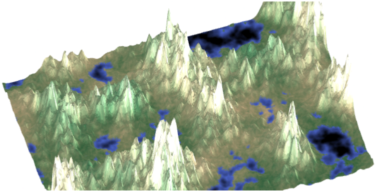
EXAMPLE

©Linda (Bohemia Interactive)

Noise Functions

Overview and Motivation

- Critical for realistic textures and models
- Simplifies creation of natural variations
- Applications: terrain, procedural texturing, simulations
- Key for realism in visual effects and games

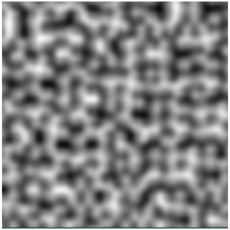


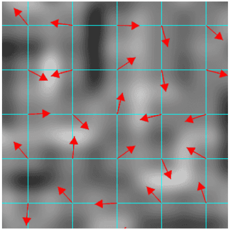
Noise Functions

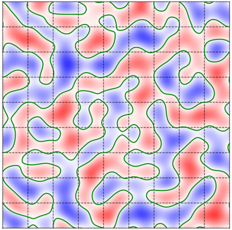
- Generate pseudo-random
- Smooth gradients – frequency limited
- Controlled randomness mimics natural forms
- Types:
  - Value
  - Gradient (Perlin, Simplex)
  - Cellular (Worley)
  - Fractal Noise

Perlin Noise

- Developed by Ken Perlin, 1983
- Algorithm:
  - Gradient vectors computed at grid points
  - Interpolated across grid to produce smooth transitions
- Properties:
  - Visually isotropic in 2D and 3D
  - Repeats over large scales, which can be controlled
- Applications: Terrain, clouds, fire textures





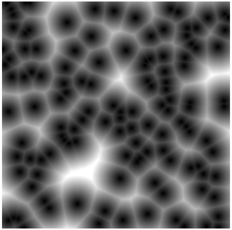


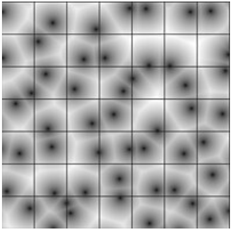
Simplex Noise

- Ken Perlin, 2001
- Algorithm:
  - Similar to Perlin but with simplex grid (triangular/hexagonal)
  - Reduces computational complexity, especially in higher dimensions
- Properties:
  - Faster computation and lower complexity than Perlin
  - Scales more efficiently to higher dimensions (4D and beyond)
- Avoids square-grid artifacts of Perlin noise

Worley Noise

- Steven Worley, 1996
- Algorithm:
  - Points randomly distributed, partitioned into cells
  - Noise generated based on proximity to nearest points
- Properties:
  - Produces a voronoi diagram-like appearance
  - Can simulate phenomena like cracked surfaces, sponge textures
- Applications: Stone, water effects, organic textures





Compositing Noise Functions

- Combines multiple noise types to increase texture complexity
- Techniques:
  - Layering different scales and amplitudes
  - Masking layers to control influence areas
- Example: Mix Perlin (base texture) + Worley (detail enhancement)
- Enhances detail and realism in procedural content

Volumetric Effects

Volumetric Effects

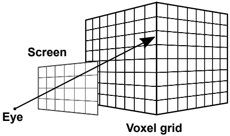
- Light usually passes through some medium (air, water, ...)
- Intensity, color (polarization) may be modulated:
  - Attenuation (fog)
  - Scattering (sunbeams, blue sky)
- Simulated by:
  - Ray traversal
  - Blending billboard slice planes

Ray Casting

- Space traversal along light ray
- Integrating properties along the ray:

$$v = \int_{ray_{start}}^{ray_{end}} f(s)ds$$

- Discrete samples:
  - Regular voxel grid
  - Procedural description
- Numerical integration:
  - Piece-wise constant
  - Interpolation (linear, polynomial)
  - …



SUNBEAMS

- Also known as crepuscular rays, god rays, ...
- Scattering on particles under direct light:
  - Sun + clouds
  - Point light source + dusty room

SUNBEAMS – IMPLEMENTATION

- Deffered shading
- Ray casting from viewer to each pixel
  - Ray sampling
  - Check if sample illuminated – shadow map test
  - Apply light scattering (physical model) to illuminated points
  - Aggregate the effect and apply to color buffer
- Heavy computation
  - Downsampled g-buffer
  - Bluring result to prevent aliasing

Other Approaches

- Create light volume geometry from shadow map and light source
  - Solve the rendering integral in intervals defined by light mesh

- Screen space approach:
  - Directional light source bluring (decreasing alpha)
  - Ligth source must be in the image