
SIGGRAPH2005

SIGGRAPH2005

Building scalable 3D applications

Ville Miettinen
Hybrid Graphics

What's going to happen... (1/2)

- Mass market: 3D apps will become a huge success on low-end and mid-tier cell phones
  - Retro-gaming
  - New game genres taking into account special characteristics of cell phones
  - Navigation apps, screen savers, animations
  - Mass market is the place for revolutions

What's going to happen... (2/2)

- Separate market for high-end game phones
  - Console ports
  - Still need to run popular ”low-end” games
  - Place for evolution
- Some devices will form their own markets
  - Launch titles still subsidized by device manufacturers
- Much more variety than in PCs or consoles

What is a ”mobile platform”?

- CPU speed and available memory varies
  - Current range ~30Mhz - 600MHz, no FPUs
- Portability issues
  - Different CPUs, OSes, Java VMs, C compilers, ...
- Different resolutions
  - QCIF (176x144) to VGA (640x480), antialiasing on higher-end devices
  - 4-8 bits per color channel (12-32 bpp)

Graphics capabilities

- General-purpose multimedia hardware
  - Pure software renderers (all CPU & integer ALU)
  - Software + DSP / WMMX / FPU / VFPU
  - Multimedia accelerators
- Dedicated 3D hardware
  - Software T&L + HW tri setup / rasterization
  - Full HW
- Performance: 50K – 2M tris, 1M – 100M pixels

Dealing with diversity

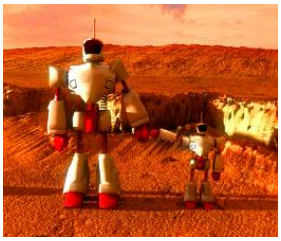
- Problem: running the same game on 100+ different devices
  - Same gameplay but can scale video and audio
- Scalability must be built into game design
- Profile-based approach

3D content is easy to scale

- Separate low and high poly 3D models
- Different texture resolutions and compressed formats
- Scaling down special effects not critical to game play (particle systems, shadows)
  - Important to realize what is a ”special effect”
- Rendering quality controls
  - Texture filtering, perspective correction, blend functions, multi-texturing, antialiasing

Building scalable 3D apps

- OpenGL ES standardizes the API and behavior
  - ES does not attempt to standardize performance
  - Two out of three ain't bad
- Differences between SW/HW configurations
  - Trade-off between flexibility and performance
  - Synchronization issues



Building scalable 3D apps

- Scale upwards, not downwards
  - Bad experiences of retro-fitting HW titles to SW
  - Test during development on lowest-end platform
- Both programmers and artists need education
  - Artists can deal with almost anything as long as they know the rules...
  - And when they don't, just force them (automatic checking in art pipeline)

Reducing state changes

- Don’t mix 2D and 3D calls!!!
  - Situation may become better in the future, though...
- Unnecessary state changes root of all evil
  - Avoid changes affecting the vertex pipeline
  - Avoid changes to the pixel pipeline
  - Avoid changing textures

”Shaders”

- Combine state changes into blocks (”shaders”)
  - Minimize number of shaders per frame
  - Typical application needs only 3-10 ”pixel shaders”
    - Different 3-10 shaders in every application
    - Enforce this in artists’ tool chain
- Sort objects by shaders every frame
  - Split objects based on shaders

Complexity of shaders

- Software rendering: Important to keep shaders as simple as possible
  - Do even if introduces additional state changes
  - Example: turn off fog & depth buffering when rendering overlays
- Hardware rendering: Usually more important to keep number of changes small

Of models and stripping

- Use buffer objects of ES 1.1
  - Only models changed manually every frame need vertex pointers
  - Many LOD schemes can be done just by changing index buffers
- Keep data formats short and simple
  - Better cache coherence, less memory used

Triangle data (1/2)

- Minimize number of rendering calls
  - Trade-off between the number of render calls & culling efficiency
  - Combine strips using degenerate triangles
  - Understanding vertex caching
    - Automatically optimize vertex access order
    - Triangle lists better than their reputation

Triangle data (2/2)

- Optimize data in your art pipeline (exporters)
  - Welding vertices with same attributes (with tolerance)
    - Vertices/triangle ratio in good data 0.7-1.0
  - Give artists plenty of automatic feedback

Transformations and matrices

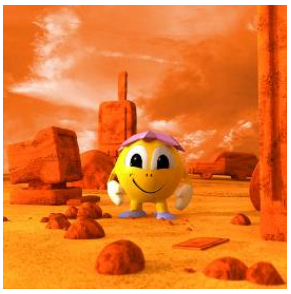
- Minimize matrix changes (demo)
  - Changing a matrix may involve many hidden costs
  - Combine simple objects with same transformation
  - Flatten and cache transformation hierarchies
- ES 1.1: Skinning using matrix palettes (demo)
  - CPU doesn't have to touch vertex data
  - Characters, natural motion: grass, trees, waves

Point sprites

- ES 1.1: Point sprites (demo)
  - Smoke, fire, explosions, water flow, stars, weather effects
  - Scale controls through PointSizeArray, PointSizeAttenuation
  - Expensive to do in ES 1.0
  - Drawback: can’t rotate sprites or textures, fixed texture coordinates

Lighting and materials

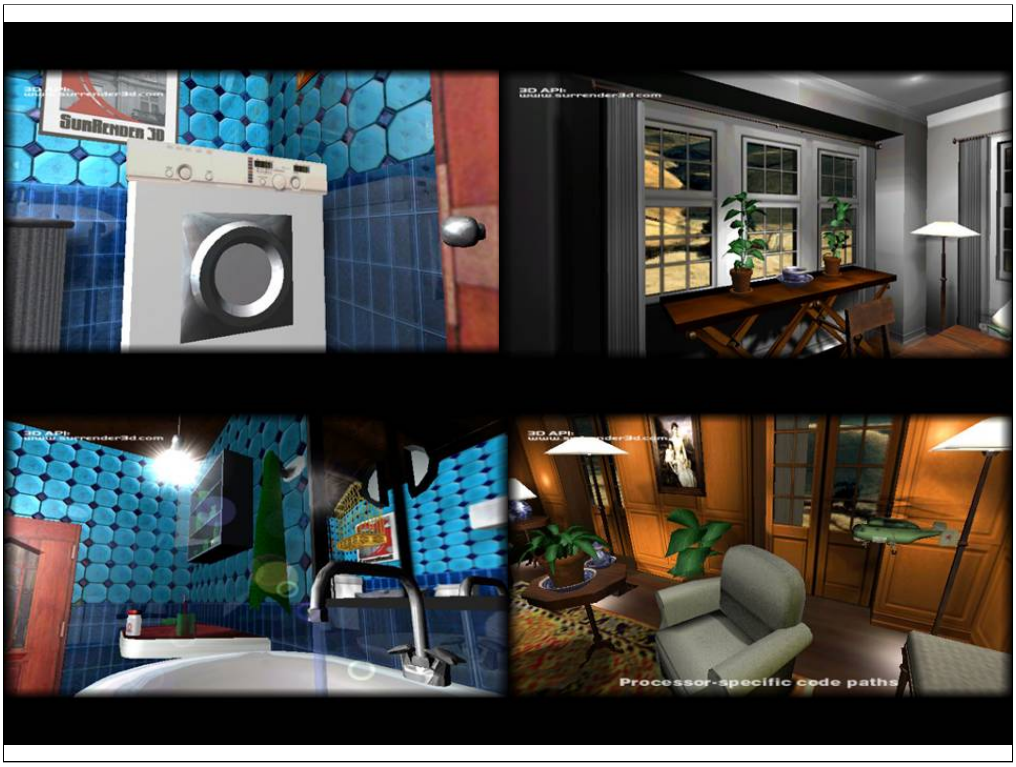
- Fixed-function lighting pipelines are so 1990s
  - Drivers implemented badly even in desktop space
  - In practice only single directional light fast
  - OpenGL's attenuation model difficult to use
  - Spot cutoff and specular model cause aliasing
  - No secondary specular color



Lighting: the fast way

- While we're waiting for OpenGL ES 2.0...
  - Pre-computed vertex illumination good if slow T&L
  - Illumination using texturing
    - Light mapping
    - ES 1.1: dot3 bump mapping + texture combine
    - Less tessellation required
- Color material tracking for changing materials
- Flat shading is for flat models!

Illumination using multitexturing


Processor-specific code paths

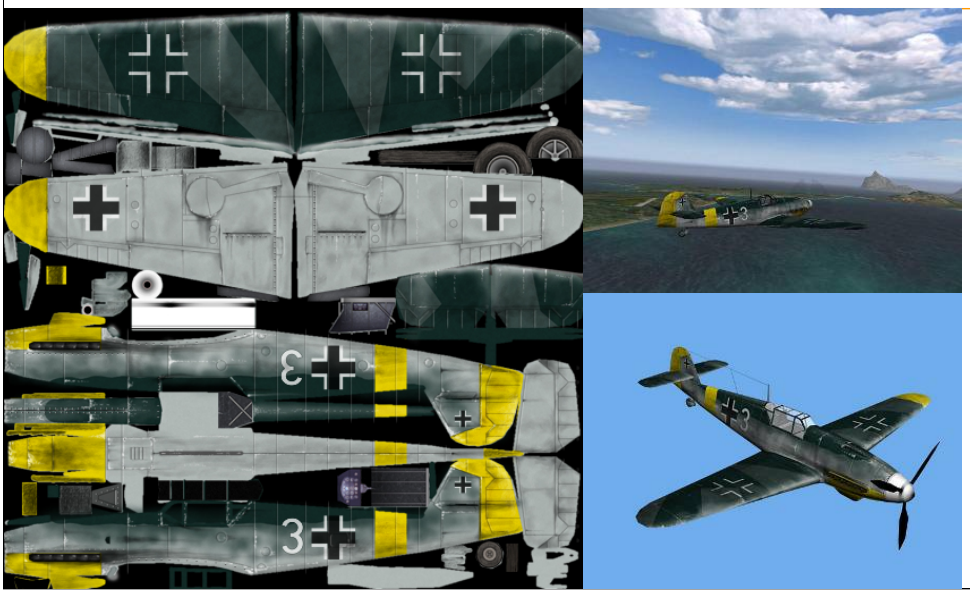
Textures

- Mipmaps always a Good Thing™
  - Improved cache coherence and visual quality
  - ES 1.1 supports auto mipmap generation
- Different strategies for texture filtering
- SW: Perspective correction not always needed
- Avoid modifying texture data
- Keep textures ”right size”, use compressed textures

Textures

- Multitexturing
  - Needed for texture-based lighting
  - Always faster than doing multiple rendering passes
  - ES 1.1: support at least two texturing units
  - ES 1.1: TexEnvCombine neat toy
- Combine multiple textures into single larger one
  - Reduce texture state changes
    (for fonts, animations, light maps)

SIGGRAPH2005



The high-level pipeline

- High-level optimizations equally important
- Setup: organize objects hierarchically
  - Triangles organized into spatially coherent ”objects”
  - Conservative bounding volumes (spheres, boxes) computed for each object

Four-step program for fast rendering

- 1. Render background and very distance objects
  - Sky cubes, impostors
    (use sky box to clear the background)
- 2. Cull objects not contributing to final image
- 3. Apply level of detail computations
  - Cull-away sub-pixel-size objects (contribution culling)
- 4. Sort remaining objects into optimal order

Culling

- Occlusion culling
  - Potentially Visible Sets and Portals are good low-cost solutions
  - ES 1.1 provides user clip planes to help with portals
- Hierarchical view frustum culling
- Back-face culling
  - Are we inside our outside the object? Terrains and indoors don't cull well

Object ordering

- Sort objects into optimal rendering order
  - Minimize shader changes
  - Keep objects in front-to-back order
    - Improves Z-buffering efficiency
  - Satisfying both goals: bucketize objects by shader, sort buckets by Z

Thank you!

- Any questions?