3D graphic acceleration – history and architecture

© 2003-2017 Josef Pelikán, Jan Horáček
CGG MFF UK Praha

pepca@cgg.mff.cuni.cz
http://cgg.mff.cuni.cz/~pepca/
Advances in computer hardware

- **3D acceleration** common in the consumer sector

  *games, multimedia*

  - **appearance**: presentation quality, ~photorealistic
    - sophisticated texturing and shading techniques, multi-pass methods, ..

- **very high performance**
  - recent **VLSI technology** *(NVIDIA Pascal .. 14-16 nm, AMD .. 1024-bit HBM memory, ..)*
  - extreme memory performance (stacked memory), **massive parallelism**
  - very fast **CPU-GPU buses** *(NVLink..)*
Advances in GPU software

- **two main APIs** for 3D graphics
  - OpenGL (SGI, open standard, Khronos)
  - Direct3D (Microsoft)

- parameter setup + **efficient data transfer**
  - sharing of data arrays (“buffers”)

- **programmable rendering pipeline**
  - revolution in realtime 3D graphics (~2000)
  - **vertex-shader**: vertex processing
  - **tessellation and geometry shaders**: geometry processing on the GPU (new primitives “on the fly”)
  - **fragment-shader** (**pixel-shader**): pixel appearance
Development tools

- for developers and artists
- high level shader programming
  - [Cg (NVIDIA)], **HLSL** (DirectX), **GLSL** (OpenGL)
  - Cg is almost equal to **HLSL**
- effect composition
  - compact definition of the effect (GPU programs, data references, parameters..) in one source file/script
  - DirectX **.FX** format, NVIDIA **CgFX** format
  - tools: Effect Browser (Microsoft), **FX Composer** (NVIDIA), **RenderMonkey** (ATI)
History I: Silicon Graphics

- first commercial graphical workstations
  - hardware-accelerated graphical subsystem (depth-buffer)
  - SW tools, libraries (Iris GL, Inventor), “graphics = SGI”

- selected workstations (http://www.g-lenerz.de/)
  - 1983: **Iris 1000** – graphical terminal to VAX (Motorola 68k @ 8MHz, Geometry Engine, ..)
  - 1984: **Iris 2000** (graphical station, Clark GE), Iris 3000
  - 1986: **Professional Iris 4D** (first MIPS processors)
  - 1988: Power series, **GTX, Personal Iris 4D**
other workstations and graphical systems:

- 1991: Iris Indigo (most popular SGI workstation)
- 1992: Iris Crimson
- 1992: Indigo R4000 (64bit), RealityEngine
- 1993: Iris Indy (cheap, 3D graphics acceleration option)
- 1993: Onyx (server with RE2)
- 1993: Iris Indigo² (Extreme graphics)
- 1996: O2 (cheap workstation), InfiniteReality engine
- 1997: Octane (2 CPUs)
- 2000: Octane2 (Vpro graphics)
History III: Consumer sector

- 1st graphic accelerator for home PC
  - 1996: 3Dfx Voodoo 1
  - graphic coprocessor ("pass-through")
  - Glide API

- 1st SLI card (Scan Line Interleave)
  - 1998: 3Dfx Voodoo²

- NVIDIA
  - 1997: NVIDIA Riva 128
  - 1998: NVIDIA Riva TNT ("TwiNTexel")

- 1st HW T&L ("transform & lighting")
  - 1999: NVIDIA GeForce 256
History IV: Consumer sector

- **2000**
  - NVIDIA GeForce2
  - ATI Radeon

- **2001: GPU programming**
  - DirectX 8.0 (vertex shaders, fragment shaders, 1.0, 1.1)
  - NVIDIA GeForce3, GeForce3 Titanium
  - DirectX 8.1 (PS 1.2, 1.3, 1.4)
  - ATI Radeon **8500** (TruForm)

- **2002: advanced GPU programming**
  - DirectX 9.0 – VS, PS 2.0
  - NVIDIA GeForce4 Titanium
  - ATI Radeon **9000, 9700** [Pro]
History V: Consumer sector

- **2003: affordable DX9**
  - cheap **DirectX 9.0** – compatible cards (VS, PS 2.0)
  - NVIDIA GeForce FX 5200-5800
  - ATI Radeon **9800**

- **2004: extended shader programming**
  - **DirectX 9.0c** (VS, PS 3.0), **OpenGL 2.0** (at last!)
  - NVIDIA GeForce **6800**, 6200, 6600
  - ATI Radeon **X800**

- **2005: HW advances**
  - **PCI-Express** bus
  - twin GPU systems – NVIDIA: **SLI**, ATI: **CrossFire**
  - NVIDIA GeForce **7800**
  - ATI Radeon **X550, X850**
History VI: Consumer sector

2006
- DirectX 10 (Windows Vista) .. geometry shaders
- NVIDIA GeForce 7600, 7900
- ATI Radeon X1800, X1900

2007
- CUDA (NVIDIA) – GPGPU programming in C
- NVIDIA GeForce 8600, 8800
- ATI Radeon R600 (HD 2400, 3850)

2009
- OpenGL 3.2, DirectX 11
- GPU tessellation
- NVIDIA Fermi
History VII: Consumer sector

- **2010**
  - **OpenGL 4**
  - **OpenCL**: general computing on GPU, multiplatform

- **2011 -**
  - **computing servers** using many GPU cards (**NVIDIA Tesla** architecture)
  - **OpenGL 4.6**
  - **DirectX 12** (Windows 10, Xbox One)
  - **OpenGL ES** for mobile platforms (**GLES 3.2**)

- **other manufacturers**
  - Past: Matrox, 3DLabs, S3, PowerVR (Kyro), SiS
  - **Intel**: very good integrated GPUs
3D graphics pipeline I

- **Application**
  - **3D data representation** (virtualization, disk + memory), parametrization, templates, ..
  - **object behavior**: physical simulation, AI
  - **interaction**: collisions, deformations, ..
3D graphics pipeline II

**Geometry** ("HW T&L")
- **modeling transforms** (application support)
- **projection transforms** (perspective), **clipping**
- **tessellation** (creating primitives “on the fly”)
- **lighting** (at least pre-computing of data for lighting)
- **very long pipeline**
### 3D graphics pipeline III

**Rasterization** (raster rendering)
- primitives are converted into **fragments**, attribute interpolation
- visibility ("depth-buffer"), **texture mapping**, shading effects, transparency, fog, ..
- **parallelism** (independent processing of fragments)
OpenGL (FFP scheme)

- OpenGL Fixed Functionality Pipeline:

```
OpenGL Fixed Functionality Pipeline:

- Vertices:
  - Geom. Operations
  - Primitive Assembly
  - Clip Project Viewport Cull

- Pixels:
  - Pixel Unpack
  - Pixel Transfer
  - Texture Memory
  - Read Control

- Frames:
  - Frame Buffer Operations
  - Per-Fragment Operations

- Pixel Groups:
  - Pixel Groups

- App Memory

- Mem. Pixels
```
OpenGL: geometric primitives

- types of geometric primitives:
  - **point**, line, polyline, closed polyline
  - polygon, **triangle**, triangle strip, triangle fan, quadrangle, quad strip

- **immediate** vertex processing **mode**
  - glVertex, glColor, glNormal, glTexCoord, ...
  - not efficient (many gl*() calls)

- **vertex arrays**
  - glDrawArrays, glMultiDrawArrays, glDrawElements, ...
  - glColorPointer, glVertexPointer, ... or **interleaving**
Geometric primitives I

<table>
<thead>
<tr>
<th>GL_POINTS</th>
<th>GL_LINES</th>
<th>GL_LINE_STRIP</th>
<th>GL_LINE_LOOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V_0 )</td>
<td>( V_0 )</td>
<td>( V_0 )</td>
<td>( V_0 )</td>
</tr>
<tr>
<td></td>
<td>( V_1 )</td>
<td>( V_1 )</td>
<td>( V_1 )</td>
</tr>
<tr>
<td>( V_2 )</td>
<td></td>
<td>( V_2 )</td>
<td>( V_2 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( V_3 )</td>
<td>( V_3 )</td>
</tr>
<tr>
<td>( V_4 )</td>
<td></td>
<td></td>
<td>( V_4 )</td>
</tr>
<tr>
<td>( V_5 )</td>
<td></td>
<td></td>
<td>( V_5 )</td>
</tr>
</tbody>
</table>
Geometric primitives II

GL_TRIANGLES

\[ V_0 \quad V_1 \quad V_2 \quad V_3 \quad V_4 \quad V_5 \]

GL_TRIANGLE_STRIP

\[ V_0 \quad V_1 \quad V_2 \quad V_3 \quad V_5 \quad V_4 \]

GL_TRIANGLE_FAN

\[ V_6 \quad V_0 \quad V_1 \quad V_2 \quad V_3 \quad V_4 \quad V_5 \]

GPU architecture 2017

Geometric primitives III

GL_QUADS

GL_QUAD_STRIP

GL_POLYGON
OpenGL “macros” (Display Lists)

- DISPLAY_LIST_MODE instead of IMMEDIATE_MODE

- sequence of OpenGL commands stored in memory
  - glNewList, glEndList
  - a list can be stored on the server side (GPU)
  - idea: “display-list = macro”

- replay
  - glCallList, glCallLists
  - could be much more efficient (sequence caching/optimization..)
Geometric data on the server

- since **OpenGL 1.5**
  - **VBO buffer**, currently: **VAO buffer**

- server-side buffers for geometric data
  - buffer management: `glCreateBuffers`, `glBindBuffer`
  - data submit: `glBufferData`, `glBufferSubData`
  - buffer mapping: `glMapBuffer`, `glUnmap..`

- working with **client memory** or **VBO buffer**
  - `glColorPointer`, `glNormalPointer`, `glVertexPointer`, ...
**Vertex Buffer Objects**

```cpp
glBindBuffer( GL_ARRAY_BUFFER, 0 );
glVertexPointer( ... );  glNormalPointer( ... );  ...
```

```cpp
glBindBuffer( GL_ELEMENT_ARRAY_BUFFER, 1 );
glDrawElements( GL_TRIANGLES, ... );
```

Diagram:
- **Vertex buffer**
  - `[x,y,z,w] [N_x,N_y,N_z] [s,t]`
  - `[x,y,z,w] [N_x,N_y,N_z] [s,t]`
  - `...`

- **VBO buffer objects**
  - `id=0`
  - `id=1`

- **Index buffer**
  - `[0,1,2] [1,2,3] [3,0,12] ...

- **GPU memory**
Vertex processing (GL < 3.x)

- **matrix transforms** (model, view, projection matrices)
  - `glMatrixMode`
  - `glLoadIdentity`, `glLoadMatrix`, `glMultMatrix`
  - `glRotate`, `glScale`, `glTranslate`, ...

- **lighting attributes**
  - `glLight`, `glLightModel`, `glMaterial`
Primitive assembly

- primitive assembly
  - how many vertices the primitive needs
  - assembly and dispatch of the data

- primitive processing
  - clipping
  - projection into a frustum, division by “w”
  - projection and clipping into a 2D window (“viewport”)
  - optional back-face culling
    - single- vs. double-sided triangles
Rasterization, fragments

- **rasterization**
  - decomposition of a primitive into a set of **fragments**
  - objects: points, line segments, triangles, bitmaps

- **fragment**
  - **raster element**, potentially contributing to a pixel color
  - size: equal or smaller than a pixel (anti-aliasing)
  - “data packet” passing through a raster part of a GPU:
    - **input/output**: x, y, z (only depth is mutable)
    - optional texture coordinates $t_0$ to $t_n$
    - specular & diffuse color, fog coefficient, user data, ...
    - **output**: RGB color and opacity $\alpha$ (further frame-buffer op.)
Fragment data interpolation

- fragment attributes are **interpolated from values in vertices**:
  - depth (z or w)
  - texture coordinates
  - colors (specular, diffuse color)
  - user attributes, ...

- fast **HW interpolators**

- **perspective correct** interpolation
  - only [x, y] is changing linearly
  - other values need one division per fragment
Fragment processing

- **texturing operations**
  - very good **optimization**
  - data fetch from texture memory
  - texel interpolation
    - mip-mapping, anisotropic filtering, ...
  - multi-texturing (many operations for combining colors)
  - special effects (bump-mapping, environment mapping)

- **fog** computation
  - depends on z value

- primary & secondary color combination (diff., spec.)
Fragment utilization ("per-fragm. op.")

- **user area clipping** (glScissor)
- **transparency** rejection test (glAlphaFunc)
- **stencil test** (glStencilOp) *write*
- **depth test** (glDepthFunc) *
- **color blending** (transparency) *
- **sRGB conversion**
- **dithering** (shallow frame-buffers)
- **logical operation** (glLogicOp) *
Global frame-buffer operations

- **frame-buffers**
  - front, back, left, right, ... (double-buffering, stereo)
  - set current rendering buffer (glDrawBuffer)

- **buffer initialization** (glClear)
  - glClearColor, glClearDepth, glClearStencil, glClearAccum

- **graphic server control** operations
  - glFlush: flush all intermediate buffers
  - glFinish: finish all current-context rendering
Raster images in OpenGL

Vertices


Frames

App Memory → Pixel Unpack → Pixel Transfer → Pixels → Texture Memory → Frame Buffer Operations

Textures

Pixel Groups

Geom. → Pixels → Pixel Unpack → Pixel Transfer → Texture Memory → Read Control → Frame Buffer
Raster image transfer

- **application memory → frame-buffer**
  - by rasterization (convert bitmap to fragments)
  - `glDrawPixels`, `glBitmap`

- **application memory → texture memory**
  - only using “unpacking” and “pixel transfer”
  - `glTexImage`, `glTexSubImage`

- **transfer inside GPU**
  - `glCopyPixels`: for frame-buffer[s]
  - `glCopyTexImage`, `glCopyTexSubImage`: target = texture

- **frame-buffer → application memory**
  - `glReadPixels`: used to be very slow operation (≤AGP)
Raster conversions and other op.

“pixel unpacking”
- conversion from app format to OpenGL format ("coherent stream of pixels", „group of pixels")
- \{ \text{RGB[α]} | \text{depth} | \text{stencil} \} \text{[]}\text{[]}
- source format, scanline length (stride), offsets, ...
- setting: glPixelStore

“pixel packing”
- OpenGL format → app format

“pixel transfer”
- change: scale, intensity, general “LuT” operation
- setting: glPixelTransfer, glPixelMap
- extensions: convolution, other filters, histograms, ...
OpenGL (Programmable Pipeline)
Vertex processor

- replaces the **vertex processing unit** in FFP
  - vertex coordinate transform
  - normal vector transform and normalization
  - computing/transformation of texturing coordinates
  - lighting vectors
  - setup of material attributes

- cannot modify
  - number of vertices
    - partial solution: primitive degeneration
  - type / topology of geometric primitives
Fragment processor

- replaces **fragment processing unit** in FFP
  - arbitrary arithmetic on fragment attributes
  - texture data fetch and application (color, etc.)
  - fog computation
  - output fragment color synthesis
  - fragment depth can be modified

- **cannot modify**
  - number of fragments (except for the “discard” operation)
  - fragment position within the viewport \([x,y]\)
Recent innovations (2009–2010)

- two more geometry processing steps on a GPU
  - geometry shader (OpenGL 3.2)
  - tessellation shaders (OpenGL 4.0)
Geometric primitives IV

GL_LINES_ADJACENCY

GL_LINE_STRIP_ADJACENCY
Geometric primitives V

GL_TRIANGLES_ADJACENCY

GL_TRIANGLE_STRIP_ADJACENCY
Geometric processors

“Tesselation shaders”
- new in OpenGL 4.0
- HW-supported surface division, subdivision (spline patches..)
- new shaders: “tessellation control“ and “tessellation evaluation“
- the former defines topology, the later actually computes new geometry (coefficients)

“Geometry shader”
- since OpenGL 3.2
- just before primitive assembly & rasterizer
- capability to create/discard vertices and primitives
- more general than TS but less efficient (unadvisable for simple [sub]division schemes)
GPU programming

- “Vertex shader”, “Fragment shader”, ...
- custom machine code executed in a vertex-, fragment-, ... processor

- application programmer is able to deploy his own code
- HW independent* programming languages
- microcode for GPU is compiled at run-time and can be effectively optimized (various profiles/versions, ..)
- low-level instructions (assembler-like code)
- or high-level languages Cg, HLSL, GLSL (similar)

NVIDIA  Microsoft  OpenGL
OpenGL 3.x

- **removed**: FFP and immediate mode (glBegin/glEnd)
- **new**: contextual profiles for future removal of obsolete functionality
- **new**: floating-point textures
- **new**: HW instancing
- **new**: geometry shaders (creating/discardng geometric primitives on the GPU)

**GLSL 1.3-1.5**
- native bitwise operations, integer arithmetic, ...
OpenGL 4.x

- **new:** HW tessellation
- **new:** direct connection to external computing API (OpenCL) w/o CPU intermediation
- **new:** binary storage of compiled shaders

**GLSL 4.0**
- 64-bit floating point computing support
- shader subroutines
- separable shader programs
Sources


- Christophe Riccino: *OpenGL reviews*, http://www.g-truc.net/post-opengl-review.html