UNC High-Performance Rendering Hardware

Anselmo Lastra
October 2001

UNC Graphics Hardware

Pixel-Planes 5
1991
fully parallel prog. shading

Pixel-Planes 4
1986
proc. per pixel 512 x 512 shadows, etc.

PixelFlow
1997
scalable shading HLL

WarpEngine
???
image-based

Pixel-Planes 1, 2, 3
1980-85
Why Is CG Difficult?

- **Floating Point Requirements**
  - Approximately 400 MFlops for 1M triangles
  - For example: Infinite Reality: Eight geometry engines at 480 MFlops each for 10 million triangles per second peak

- **Memory Bandwidth**
  - Approximately 250 million frame buffer accesses to rasterize 1 million 100-pixel triangles

Rasterization

- **Memory size increase 3-4 orders of magnitude**
- **Speeds have not kept up!**
- **Number of pins remained relatively constant.**
- **Distributed frame buffers were proposed to address the problem [Fuchs77][Parke80].**
Pixel-Planes

- Henry Fuchs’ Idea: build processing into the frame buffer, a processor per pixel.
  - UNC designs are called enhanced memories,
  - not SIMD processors
- Enabler was linear expression tree…

Linear Expressions

Ax + By + C
Linear Expressions

Ax + By + C

Linear Expressions

Ax + By + C
Depth and Color Interpolation

\[ z = F(x, y) \]
and color = \[ F(x, y) \]
for each of red, green, and blue

Pixel-Planes 1, 2, 3

- Pixel-Planes 1 - 4 processors
- Pixel-Planes 2 - 4 by 64
- Pixel-Planes 3 - 64 by 64
Pixel-Planes 4 (1986)

- Full-size (512 by 512 pixel) prototype
- 2048 enhanced memory ICs
- One Geometry Processor
- 72 bits memory per pixel

Pxpl4 Block Diagram

- 512 x 512 frame buffer
- 128 processors/chip
- 2048 EMC chips
- 32 boards
- Clock speed: ~ 10MHz
### Performance

- 35K triangles per second
- Spheres as a primitive
- CSG
- Shadows

### References

- The first reference describes the algorithms, while the second describes the machine as-built.
Lessons Learned from Pxpl4

- Programmability useful!
- More pixel memory required
- Tris small,
  - many processors unused
- Must extend parallelism to geometry processing → fully parallel pipeline

Graphics Pipeline

User-Specified Coordinates (floating pt.)
(3.7, 0.9) (3.8, 0.4) (4.1, 0.5)

Screen Coordinates (integer)
(273, 407) (240, 390) (296, 396)

Pixels with correct color

Geometry Processing
Rasterization
Sorting Classification

- WarpEngine
  - Geometry Processors
    - sort first
  - Pixel-level Processors
    - sort middle
- Pixel-Planes 5
  - sort middle
- PixelFlow
  - sort last

Sort Middle

- Straightforward and well known.
- Network limits scalability.
- Somewhat scalable in display size.
Slide 19
Lastra, Oct. 2001

Sort Middle (Pixel-Planes 5)

- Perform geometry processing in parallel by primitive. Sort each transformed primitive to determine where on screen it belongs.
- Route transformed primitive to renderer responsible for appropriate screen regions.

Frame Buffer

G G G G
R R R R

Application

R R R R

G G G G

Slide 20
Lastra, Oct. 2001

Screen-Space Subdivision

Pixel processors (128 x 128) remapped to screen tiles

Problem: Polygons must be sorted!
Virtual Frame Buffer

- Must sort all primitives before scan conversion
- Pipelined two frames,
  - one in geometry stage,
  - another in scan conversion
- Penalty: memory & increase in latency

Load Balancing on pxpl5

- Greedy algorithm
- Master collects info on # of prims / region
- Starts token, which flows GP to GP

Rasterization order: 1 2 3 3 2 1 2 3
Size of box indicates # of primitives

- Last GP sends token back to master

When does this fail?
Implementation

Host

Graphics Processor

160 MW/s ring network

Frame Buffer

Sun Workstation

Renderer

128 x 128

Renderer

128 x 128

Renderer

128 x 128

Renderer

128 x 128

multiple frame buffers: hires, dual NTSC, color sequen.

Technology

- 1-bit ALU
- Quadratic evaluator
  \[ Ax^2 + By^2 + Cx + Dy + Exy + F \]
- 208 bits/pixel
- 4K backing store/pixel
- 40 MHz clock speeds
Shading

- Finally enough bits to do cool shading!
- Language was assembler with macros (sqrt, norm, etc). Word length variable.

Performance

- Triangles small, but the size doesn’t matter as long as overlap factor does not increase much!

Record Performance on GPC "head" Dataset.

Sierra Nevada Elevation Dataset

Dataset courtesy Herman Towles and Sun Microsystems

Model courtesy GPC committee (now SPEC)
References


Limits to Scalability

Crossbar bandwidth must increase with number of primitives.
Image Composition

- Each node renders subset of primitives.
- Depth determines which sample proceeds.

256 wires @ 200 MHz, bi-directional (> 100 Gb/s) board to board

Sort Last (Image Composition)

- Scalable in # of primitives
- Requires high-bandwidth network (next slide)
- Anti-aliasing expensive
- Transparency difficult.
- Not scalable in display size.
Composition Bandwidth

- Need enough bandwidth for:
  \[
  \text{display size} \times \text{frame rate} \times \text{subsampling}
  \]
- For \(1280 \times 1024\) at 72 Hz, with 5 sample antialiasing,
  - need 10 Gigabits/sec of bandwidth

PixelFlow Goals

- Investigate image composition
- Add high-level shading language
  - Turned into programmability everywhere in pipeline
- Immediate mode from a parallel machine

Video
**PixelFlow Node**

- **PA-8000**
  - Memory Interface
  - 128 MB Memory
  - I/O Card (opt.)

- **PA-8000**
  - Geometry Network Interface

- **Texture Memory** 64 MB

- **Optional Video Adapter**

- **Pixel Processors** (128 x 64)

- **Composition Network**

---

**Shading Functions**

- **Pins**
  - Crown, label, scuffs, dirt, Phong

- **Alley**
  - Wood, reflection map

- **Ball**
  - Phong

- **Light**
  - Shadow map

*Image by Yulan Wang, UNC.*
Bump Maps

Images by Brad Ritter, Hewlett-Packard.

Enabling Programmable Shading

• Shading Language
  – pfman
  – Similar to RenderMan with extensions mainly for speed on PixelFlow
• Extensions to OpenGL to allow access from geometry code.

Videos
#define BRICK_WIDTH 0.25
#define BRICK_HEIGHT 0.08
#define MORTAR 0.01

surface brick(
    output unsigned varying fixed<8,8> gl_rc_co[3],
    unsigned texture varying fixed<16,16> gl_material_texcoord[2]) {

    float ss, tt;
    fixed<8,0> row;

    tt = gl_material_texcoord[1] % (BRICK_HEIGHT + MORTAR);
    row = gl_material_texcoord[1] / (BRICK_HEIGHT + MORTAR);

    gl_rc_co[0] = 0.5; // both brick & mortar same red
    if(tt > brick_height) {
        gl_rc_co[1] = tt > brick_height ? 0.5 : // within mortar row
            0.5; // within brick row
        gl_rc_co[2] = 0.5;
    }
    else {
        ... // within brick row
    }
}

Deferred Shading

Rasterization Nodes

Geometry Processor

Geometry Processor

Rasterizer

Rasterizer

Shading Node

Geometry Processor

Rasterizer

Appearance Parameters (parameters necessary to shade the image, such as normals, etc.)

Color to Frame Buffer (another Pixmap node)

• User selects the number of rasterization and shading nodes to suit the workload.
**Advantages and Disadvantages**

- You only shade pixels that are visible
- Increased coherence
  - but...
- Must save and transmit many parameters → high bandwidth demand

**Transparency**

- A problem with image composition
- We implemented
  - Screen door when antialiasing
  - Mammen’s algorithm with transparent polygons on shading nodes
- Still, sort-last is not good when many polygons are transparent
Immediate vs. Retained Mode
(direct vs. indirect rendering)

- Immediate-mode limited by host to graphics bandwidth.
- Display lists stored on nodes
- Our intent was to attach PixelFlow to a parallel machine

Performance

- PixelFlow was built with industrial partners, first Division, then HP
- Demonstrated running at about 43 million triangles per second on 36 nodes
References


Why Use Polygons At All?

• Pros
  – Convenient for modeling (by hand)
  – Good representation when large on screen
  – Useful for man made objects
• Cons
  – Huge number to model natural scenes
  – Fairly complex to render

WarpEngine

• Hardware architecture for rendering from depth images

Voicu Popescu
  – Also John Eyles,
  – Josh Steinhurst

Kamov Video
Rendering Algorithm

- WarpEngine algorithm
  - Interpolate between reference image samples
  - Warp (transform) them forward to image space
  - Z-compose into sub-pixel (2x2) warp buffer
- No interpolation across “skins”

Forward vs. Backward Map

- Conventional scan conversion
  - For each pixel, compute color
  - Basically backward map
- WarpEngine
  - Warp sample forward
Offsets Make it Work

- 2-bit offset
- More precise sample location
- 2-pixel wide filter kernel
- Similar to sparse buffer

Blue – pixel, Green – warp buffer, Black – offset

Inexpensive Antialiasing

2 x 2 Offset  No Offset

Zoomed
Why Forward Map?

- Low setup cost!
  - No edge-expression computation
- Exploits coherence
  - IBR tile (16x16 image) tends to need same interpolation factor
  - Can use efficient SIMD warper

Architecture

[Diagram of architecture showing components such as Region Accumulator, Reconstruction Buffer, Warp Array, and Network Interface connected to Tile Cache and FrameBuffer.]
WarpArray

- Nearest neighbor connectivity
- In/Out/Warp pipelined
- Similar to PixelFlow design

Region Accumulator

- Pixel interleaved
- 128 x 128
- Soft z?
- Reconstruction pipelined with next region rendering
Sort First for Parallelism

- How to distribute work across chips?
- Sort by screen space regions
  - 128x128 pixel region
- Sort First [Mueller] refers to sorting primitives as soon as possible
- Tile coherence lowers overlap factor

Sort First

- At first glance, like sort middle.
- Advantage: only primitives that move to other screen regions need to be transferred
- May scale better in display size
- Difficulties in memory access and editing
Expected Chip Specs

- ASIC 12x16 mm
- 0.18 micron
- $\geq 300$ MHz
- 4-node VGA
- 32-node HDTV
- Each chip
  - 100M Samples/sec
  - 4.8G Bytes/sec bandwidth

References

Future

- Programmable shading (finally happening!)
  - More memory
  - Floating point
  - Regular architecture?
- Take advantage of small primitives
- Support for image-based primitives?
- Big displays with lots of pixels!

Credits

Pixel-Planes 4: Greg Abram, John Austin, Fred Brooks, Vern Chi, John Eyles, Henry Fuchs, Eric Grant, Trey Greer, Jack Goldfarber, Scott Hennes, Jeff Hultquist, Mark Monger, John Poulton, Susan Spach, John Thomas
Pixel-Planes 5: Mike Bajura, Andrew Bell, Brad Bennett, Jeff Butterworth, Vern Chi, Jon Cohen, Grant Cooper, Claire Durand, David Ellsworth, Niall Emmart, John Eyles, Henry Fuchs, Howard Good, Edward "Chip" Hall, Sonya Holder, Linda Houseman, Heather Kons, Victoria Interrante, Roman Kuchukuda, Subodh Kumar, Anselmo Lastra, Jonathan Leech, Jeff Mauldin, Steven Molinar, Carl Mueller, Ulrich Neumann, Marc Olano, Sherry Palmer, Mark Parris, John Poulton, John Rhoades, Eddie Saxa, Andrei Stale, Bruce Tebbs, Steve Tell, John Thomas, Russ Tuck, Greg Turk, Sharon Walters, Laura Weaver
PixelFlow (Division and Hewlett-Packard): Byron Alcorn, Greg Allen, Peter Arnold, Phil Atkin, John Austin, Randy Bailey, Tony Barbour, Tony Bennett, Dave Bradley, John Cook, John Dykstal, Trey Greer, Charles Grimsdale, Bill Hale, Larry Harbold, Fred Heath, Justin Heinecke, Kenny Hoff, Maureen Hoffer, Rich Holloway, Mary Jo Jarowicz, Paul Keller, Kevin Koeher, Roman Kuchukuda, Paul Layne, Ray McConnell, Mike Meyers, Chris Pratico, Brad Ritter, John Rhoades, Dunward Rogers, Gary Schaffer, Doug Schiff, Rex Seader, Robert Seward, Mel Snyder, Herman Towles, Jim Weaver, Lee Westover, Robert Yockey
WarpEngine: Gary Bishop, Nick England, John Eyles, Anselmo Lastra, Lars Nyland, Voicu Popescu, Josh Steinhurst

Funding: DARPA and NSF. Additional support from Division, Intel, Hewlett Packard.