Software implementations of rendering are slow.

- OpenGL on Sparc workstations.

Performance can be improved using sophisticated algorithms and faster machines.

Real-time large-scale 3D graphics is not possible without hardware support.

At least some of the rendering steps can be replaced by hardware.

Rendering is ideal for pipeline and multiprocessor architectures.
A Brief History

**Generation:** Capabilities for which the architecture was primarily designed.

**First Generation**

- First machines came out in early 1980s.
- Transformation capabilities.
- Limited frame-buffer processing.
- Flat shading.
- Smooth shading, z-buffer not supported.

*Examples:*
  - SGI Iris 3000 (1985);
  - Apollo DN570 (1985).

*Later machines:* Limited smooth shading & depth buffering.
  - *Examples:* SGI 4DG (1986).
  - Effective for wire-frame images.
The Second Generation

★ Reduced memory costs &

*Application-specific ICs* (ASICs):
- Allowed large frame-buffer with multiple rendering processors.

★ Interpolation of colors and depths.

★ Memory capacity & bandwidth allowed depth buffering.

★ *Examples:*
- SGI GT (1988);
- Apollo DN590(1988).

★ Later machines: limited texture mapping.

★ Antialiasing of points and lines.

★ *Examples:*
- SGI VGX;
- HP VRX;
- Apollo DN1000.
The Third Generation

**SGI Reality Engine:**

- Lighting, smooth shading, depth-buffer, texture mapping, and antialiasing.
- 0.5 millions triangles per second, under assumptions:
  - triangles in short strip.
  - 10% triangles intersect the view frustum.
- Filtering for textures; large textures.
- Antialiasing for polygons.
- *Pixel fill rate:* 30Hz rendering of $1280 \times 1024$ full-screen images.

**SGI Infinite Reality:**

- Pixel fill rate 60Hz.
- Virtual texture memory.
- Display-list memory on graphics processor.
- Onyx and Onyx2 platforms.
Standard Graphics Pipeline

Display traversal → Modeling transform → Trivial accept/reject → Lighting

View Transform → Clipping → Viewport Mapping → Rasterization

Monitor
Assume polygons are being rendered.

- Application processing between frames.

- **Geometry processing**: 3D Polygons to 2D polygons (in screen coordinates).
  - Transformation from local to world coordinates.
  - Lighting at vertices.
  - Transforming to a canonical view volume.
  - Clipping.
  - Perspective projection.
  - Transformation to screen coordinates.

- **Rasterization**: 2D polygons → pixels.
  - Scan conversion.
  - Shading.
  - Hidden surface removal.

- **Display processing**: Converting pixels to analog display.

*Front end vs back end.*
Geometry processing is ideal for pipelined processing.

- Earlier stages process the next polygon while later stages are processing the current polygon.
- Latency and throughput.
- SGI used pipelined processing in early architectures.
Geometry processing faster on parallel machines.
★ Polygons can be processed in parallel.
★ Each processor performs all steps of geometry processing on a polygon.
★ Multiple polygons are processed simultaneously.
★ Recent SGI systems use this approach.

Rasterization is ideal for parallel processing.
★ $1280 \times 1024 \approx 1.3M$ pixels need to be processed per frame.
★ Supersampling: $\#$ subpixels $\approx 10$–$20$ million.
★ Most pixels are processed multiple times in $z$-buffer algorithms.
★ Use multiple processors.
Contiguous vs Interleaved

☆ Contiguous partitioning performs well in the best case.
- Polygons are uniformly distributed.
- Each processor handles only a fraction of the polygons.
- Load on each processor is balanced.

☆ Performs poorly in the worst case.
- All polygons are in a local region.
- A few processors do all the work.

☆ Interleaved is best in the worst case.
- Each processor handles all the polygons.
- Load is balanced.
SGI Reality Engine

★ Three, four, or six graphics boards.

★ **Geometry board:**
  - Input FIFO
  - Command processor
  - Geometry engines: 6, 8, or 12.

★ **Raster memory board:** 1, 2, or 4.
  - 5 fragment generators.
  - Each with its texture memory.
  - 80 image engines.
  - Each image engine with frame buffer memory: \( \geq 256 \) bits per pixel.

★ **Display board:** Video functions.
  - Video timing
  - Color mapping
  - D/A conversion

*FIFO memories* at
★ Input and output of each geometry engine.
★ Input of each fragment generator.
★ Input of each image engine.
Command Processor

- Receives OpenGL command from applications and other processors.
- Directs each triangle to one of the geometry engines.
  - Round-robin distribution.
  - No load balancing.
- Infrequent command: e.g., matrix multiplication, lighting model.
  - Broadcasted to all geometry engines.
  - Synchronization is required.
- Frequent command: e.g., vertex color, coordinate, normal.
  - Bundled with each rendering command.
  - Sent to individual geometry engines.
- Breaks long connected sequences of segments and triangles into short groups.
- Each piece sent to a single geometry engine.
To Triangle Bus

Output is sent to the triangle bus.

- Transformations
- Lighting
- Clipping
- Projection
- Texture

- Connecting all geometry engines to all fragment generators.
- 1M smooth shaded, depth buffered, texture mapped, triangles per second.
- Depth, texture calculations in double precision.
- Peak performance 100MFLOPS
Output of geometry engines is sent to 5, 10, or 20 fragment generators; say 20.

Each fragment generator responsible for 1/20 of the screen’s pixels; 64K pixels.

Interleaved partitioning of the screen.

Computes the intersection of the set of pixels fully or partially covered by the triangle.

For each fragment it computes
- Depth, color (including texture)
- A subsample mask for each fragment.

Output is sent to 16 image engines.
Fragment Generator

- Subsample mask:
  - 4, 8, or 16 samples from a $8 \times 8$ grid.
  - Same subsamples for each pixel.

- Depth is computed at the center sample.
  - Ensures accurate depth calculation at each subpixel location.

- Color sample values:
  - If triangle covers the pixel, compute color at the center.
  - If partially covers, compute near the centroid of intersection of triangle and pixel.

- Incremental algorithm for rasterization.
16 Image engines connected to each fragment generator.

Each image engine responsible for 4K pixels. (*Or* 8K, 16K *if fewer raster memory cards.*)

Each image engine assigned to a fixed subset of pixels.

Each image engine controls a 256K × 16 DRAM.

DRAM forms frame and depth buffers.
Each pixel is assigned 1024 bits
(Or 512, 256 if fewer raster memory cards.)

These bits store:

- **Color** (R, G, B, A values) for each subpixel
  - 12 bits each if 8 subsamples;
  - 8 bits each if 16 subsamples.

- **Depth**:
  - 32 bits if 8 subsamples
  - 24 bits if 16 subsamples.

- 1, 2, 4 1280 x 1024 displayable color buffers

- Displayable buffers have the same resolution as subsamples.
Image Engines

★ When a triangle $\Delta$ is passed to a fragment generator, it’s slope information is passed to image engines.

★ Using slope information, image engines compute depth at each subpixel of $8 \times 8$ grid.

★ For each 1 in the mask, depth value is compared with the value stored in the depth buffer.

★ If comparison succeeds
  
  • Color and depth values in the framebuffer are updated.
  
  • Aggregate color value is recomputed.
  
  • New color value is rewritten on the displayable buffer.
Infinite Reality
Infinite Reality

- Pixel fill rate $\geq 60$ frames per second.
- Each pixel is assigned 0.5–2K bits.
- Speed up command processing.
- Display lists transferred from the host processor using DMA transfer.
- 15MB memory to store display lists at the graphics processor.
- Customized geometry processors.

- Geometry distribution:
  - Round robin: Simple assignment.

- Vertex bus instead of triangle bus.
  - Triangle slope information is not passed.
  - Only vertex information is passed.
  - Reduces bandwidth by 60%.
  - Load on vertex and input buses are similar.

- Additional hardware for texture mapping.
Hardware Information

Ring Network

Graphics Processor

Host Interface

Frame Buffer

Renderer

Workstation
(Fuchs et al., 1989)

★ **Performance:** \( \approx 2.3 \text{M triangles per second}. \)

★ Parallel geometry processing.
  - 50 *graphics* processors

★ Parallel rasterization.
  - Contiguous partition.
  - 20 *renderes*.
  - separate shaders.
★ Receives polygons from application or other processors.

★ Performs geometric processing.

★ Assigns processed polygons to contiguous partitions.
  - Each partition is $128 \times 128$ square.
  - Each graphics processor has a bin for every partition.

★ After all polygons are processed, all bins are passed to renderers.

★ Communication is through a high bandwidth ring network.
All bins are processed in parallel

✦ A renderer rasterizes all polygons in one partition’s bin from each graphics processor.

✦ After processing these bins, renderree processes bins of another partition.

Rasterization is performed using logic-enhanced memory.

✦ A small processor for each of the $128 \times 128 = 16K$ pixels.

✦ 2K memory for each pixel.

✦ Each processor maintains a few states, e.g., its $x$- & $y$-coordinates, and evaluates

$$Ax + By + C + Dx^2 + Exy + Fy^2$$
Pixel Flow

(Molnar et al., 1992)

Being developed by Hewlett-Packard.

✫ Unbounded parallelism in theory; MIMD machine.

✫ Performance: in theory: unlimited polygons per second.

✫ Rendering is performed by $n$ complete rendering systems.
  
  • Each system includes a graphic processor, renderer, and a shader.
  
  • Each system processes $1/n$ polygons.
  
  • It outputs the frame buffer and also the $z$-buffer.

✫ Composites $n$ different images to produce the overall image.

✫ Unlike other machines, one pixel may be processed by many processors.