Real-Time Graphics Architecture

Kurt Akeley

Pat Hanrahan

http://www.graphics.stanford.edu/courses/cs448a-01-fall

The Design of RenderMan

Religious Issue #1

The world is not made of polygons

In fact, polygons weren't originally supported!

Collection of complex geometric primitives

- Fractals
- L-systems and plants
  - Proceduralism and data amplification
- Curved surfaces
  - No need to provide normals in the real world

Religious Issue #2

Realism results from detailed and diverse geometry, lighting and texturing

Visual or cinematic complexity

- Visual detail per-pixel
- Lots of flops per pixel: computational threshold (just now crossing)

Implications of diversity

- No single modeler, thus no single shader
- Difficult to offload the complexities of shading to the modeler
- Flexibility and extensibility critical

Religious Issue #3

"Don't make pictures that require apologies"
Alvy Ray Smith

Remove all computational artifacts (digital signatures)
- Aliases, quantization, …

Heavy emphasis on doing it right
- Stochastic sampling
- Resampling, minification and magnification
  "A pixel is not a little square" (hint: it's a sample)
- Adaptive subdivision (no polygonal silhouettes)

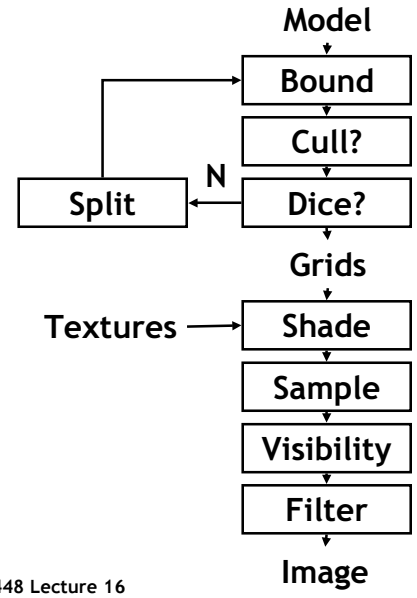
Again, sort of a threshold phenomena

REYES

Renders Everything You Ever Saw (Carpenter)

The Road to Point Reyes
Directed by R. Cook, LucasFilm 1983

REYES Pipeline



Reyes Architecture

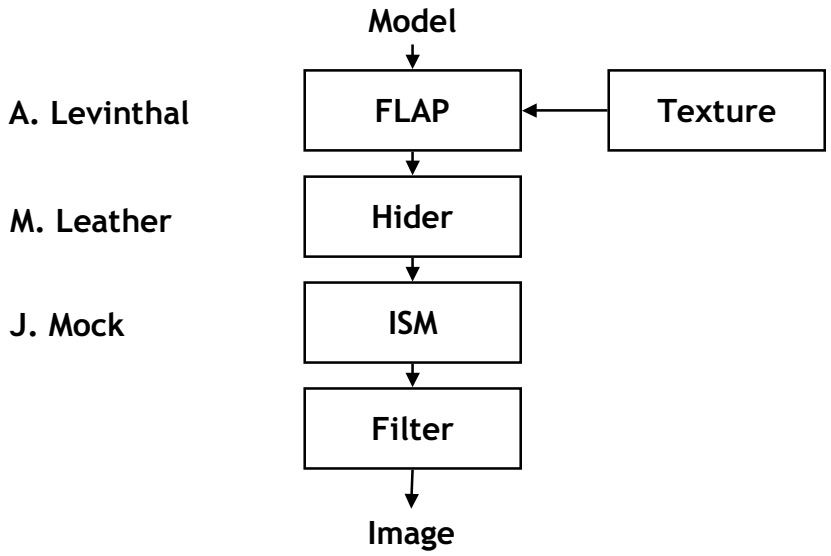
- Single internal low-level geometric primitive
  - Micropolygons (microquadrilaterals)
  - Dicing creates grids consisting of micropolygons
- Nyquist-sized
  - ¼ pixel area (½ pixel width)
  - Flat shading adequate
- Split and dice along surface parameters (u, v)
  - Coherent geometry and texture (texture in shading-space)
- Shading before hiding (hiding=rasterization + z-buffer + …)
  - Shade in object-order and in object space
  - Displacement maps (texture before hiding)

REYES Machine Goals (1986)

Micropolygons (area=¼ pixel)	80,000,000
Pixels	3000 x 1667 (5 MP)
Depth complexity	4
Samples per pixel	16
Geometric primitives	150,000
Micropolygons per grid	100
Shading flops per micropolygon	300
Textures per primitive	6
Total number of textures	100 (1 MB/textures)

Goal ~ 1 frame in 2 minutes – not real-time

REYES Machine Architecture



Influences

Postscript

Page description language for the printed page

RenderMan

Scene description language for photorealistic imagery

Design Study #1: Geometric Prims.

REYES had tens of primitives (O-O framework)

- Fractal terrain
- Tear-drop
- …

Primitives (Fournier)

- Modeling primitives: e.g. chair
- Rendering primitives: e.g. b-spline
- Display primitive: i.e. triangle or micropolygon

Design Study #1: Geometric Prims.

RenderMan (identifiable, top-down primitives)

```
RiSphere(r,h,…)
RiPolygon(nverts, "P", Ps, "N", Ns, … );
```

OpenGL (assembly-based, bottom-up primitives)

```
glBegin(GL_POLYGON);
  glNormal3fv(&N[0]);
  glVertex3fv(&P[0]);
  …
glEnd(GL_POLYGON);
```


Design Study #1: Geometric Prims.

Touch choices

- Display lists turned into macros (relented)
- PointsPolygon (relented)
- Trimmed NURBS (relented)
- `RiGeometry("teapot");` (obviously needed)

Correct decisions?

- Geometric complexity now "out of control"
- Composite objects (e.g. trees): LOD not solved
- Procedural primitives: lazy evaluation
- Scanned primitives: not widely used yet

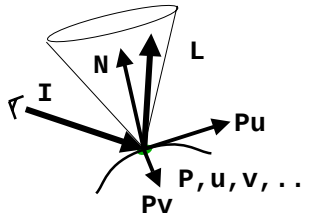
Design Study #2: Lighting

ShadeTrees:

```
Ci = Cd*diffuse(N)+Cs*specular(H,s);
```

Extensible BRDF models

```
illuminance(N, Pi/2) {
  Ci += Cd*Cl*max(N.L);
}
```



Point and area lights

- Single `illuminate` construct (not right!)

Environment maps should be treatable as light sources

- `solar` construct

Design Study #2: Lighting

Domain-specific or "little" languages should provide builtin functions and operators that are high-level and make it easy to use

Just as important, builtins provide efficiency

- Large calculation quanta allow you to amortize the overhead of the "interpreter"
- Allow superoptimized implementations

But, builtins should be expressable in the language!

"Programming languages should be designed not by piling feature on top of feature, but by removing the weaknesses and restrictions that make additional features appear necessary"

IEEE Scheme Standard

Basic Design Cycle

Basic design cycle

- Propose examples
- Propose interface
- Express the examples using the interface
- Iterate, simplifying and enhancing

Lightweight design cycles

- Minimal specifications at first (Quick-Spec)
- Prototype to make sure it works

Abstractions

Build your system around abstractions

- CS101: hide details of the implementation

Identify the properties of the abstraction (semantics!)

- This allows composability (mechanism)
- And compilability and optimizability

Strong vs. weak abstractions

- Color and point types vs. vector type

Abstractions have a cost and a benefit

- Strong typing is good
- Restrictions come at a cost

Abstractions ultimately limit the system

- Light and scene abstraction vs. stroke abstraction

Implications of the Shading Language

The interface is simplified

- RenderMan ~ 100 API calls, OpenGL ~ 300 API calls
- Examples:
  - Nothing like two-sided lighting
  - No fragment calculations

The implementation is much more complicated

- 90% of the code
- Investment in the future

Advantages

- Original goal: Provide high quality shading models
- Extensibility, adaptability, and efficiency!
- Results in relatively stable interface

Things I'm Most Proud Of …

Size and stability of the interface

Shading language

- `illuminance` and `illuminate` constructs
- Derivatives
- Texturing primitives
- `uniform` and `varying` variables

Mechanisms for controlling level of detail

Procedural primitive interface

Camera model (not just view)

Things I Might Change Now …

- Support for spectral computations (rarely used)
- Vertex variables (too complicated and just plain ugly)
- Derivative operator
  - Replace with mipd=area() with derivatives … cool, general
  - Difficult for ray-tracers and hardware to implement
- Deformations
  - Elegant, just add a programmable transformation
  - But, difficult to adaptively tesselate and bound
  - Lesson: we should have implemented this first!
- Linear integrate and BRDF shaders
  - Didn't appreciate the subtleties of specifying reflection
  - Linearity is a big issue in lighting design
- More sophisticated camera model (lens flare, tone repr.)

Things I Never Figured Out …

And hence, punted on …

- Integration of 2D and 3D and 4D(t) graphics
  - How to integrate seamlessly?

  The achilles heel of I3D graphics
- Material properties when doing CSG
  - What happens when you subtract a wood sphere from a metal cylinder?
- Curve and edge, point shaders
  - What is the normal to a curve?
- Procedural file format (ala PostScript)
- Metarenderer

Things I Didn't Know About R-T. R. …

Only after I went to Princeton and used GL did I appreciate many of these things …

- Connection to the application

  Power of immediate mode graphics (e.g. LOD)

  Access to application data structures

  Performance issues in the interface
  - Naming and binding mechanisms
  - Low-level data types (2f, 3f, 4fv, …)
  - Think AGP and networks …
- Range of interactive applications

  Image processing and warping via texturing

  Integration with GUI and window systems

Things I Was Clueless About …

Physically-based rendering

- Radiosity, radiance and BRDFs

Semantics of programming languages

- Functional programming languages
- Type inference

And, it shows …

Things Others Have Added …

Subdivision surfaces

Smart lights

Enhancements to procedural primitives

How to Design

Lightweight design cycles

Be precise

- Make sure foundational parts perfect
- Don't hide behind imprecise thinking, specify!
- Identify the tricky cases; provide torture tests
- Formalize, but beware of "overconcreteness"

Have a sounding board

- Einstein's assistant at the IAS

Talk to the real users

- Don't live in an ivory tower
- But beware of NIH, lack of understanding of the technology, and lack of vision

Study and learn from other good designs

Being an Architect

No silver bullet, F. Brooks

Hints on system design, B. Lampson

Dealing with management

- Do they appreciate good design?

Dealing with your colleagues

- Need to know everything, so learn from everyone
- Diplomacy and communication skills essential
- Be rational and fair
- Expect to take a lot of flak

Agreement with Kurt's talk

Beauty counts

Importance of specification

Don't design by committee (industry standards)

Co-development of implementation and interface

At least 2 target platforms

…