CMSC 635

Procedural Shading

Idea of Shading

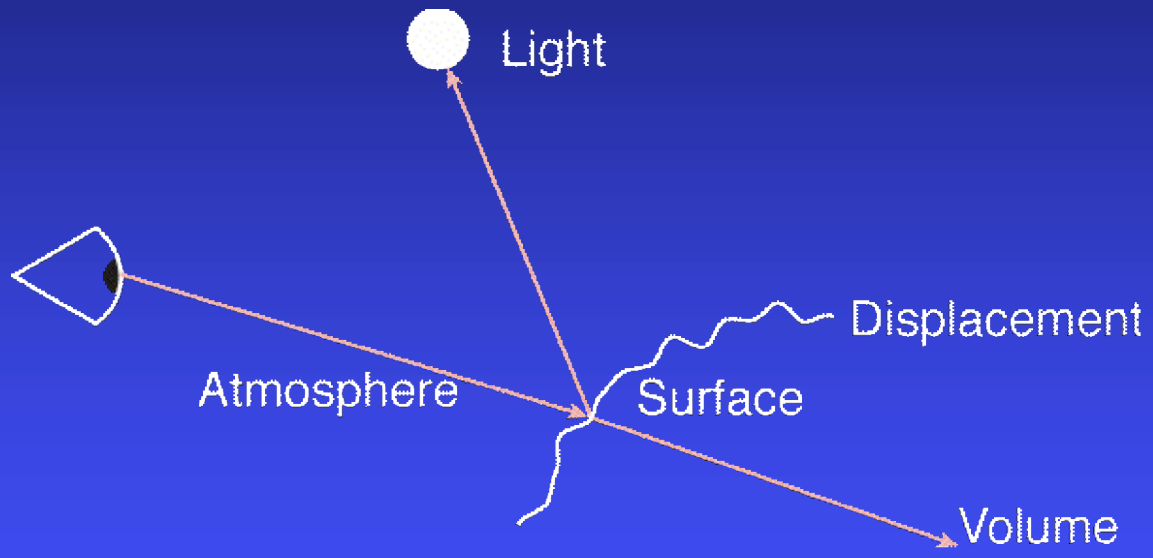
- Want more flexibility
- Procedures for aspects of appearance
  - Compute base surface color
  - Interaction with light
  - Color of light
  - Attenuation through space
  - Fine-scale surface features

Examples

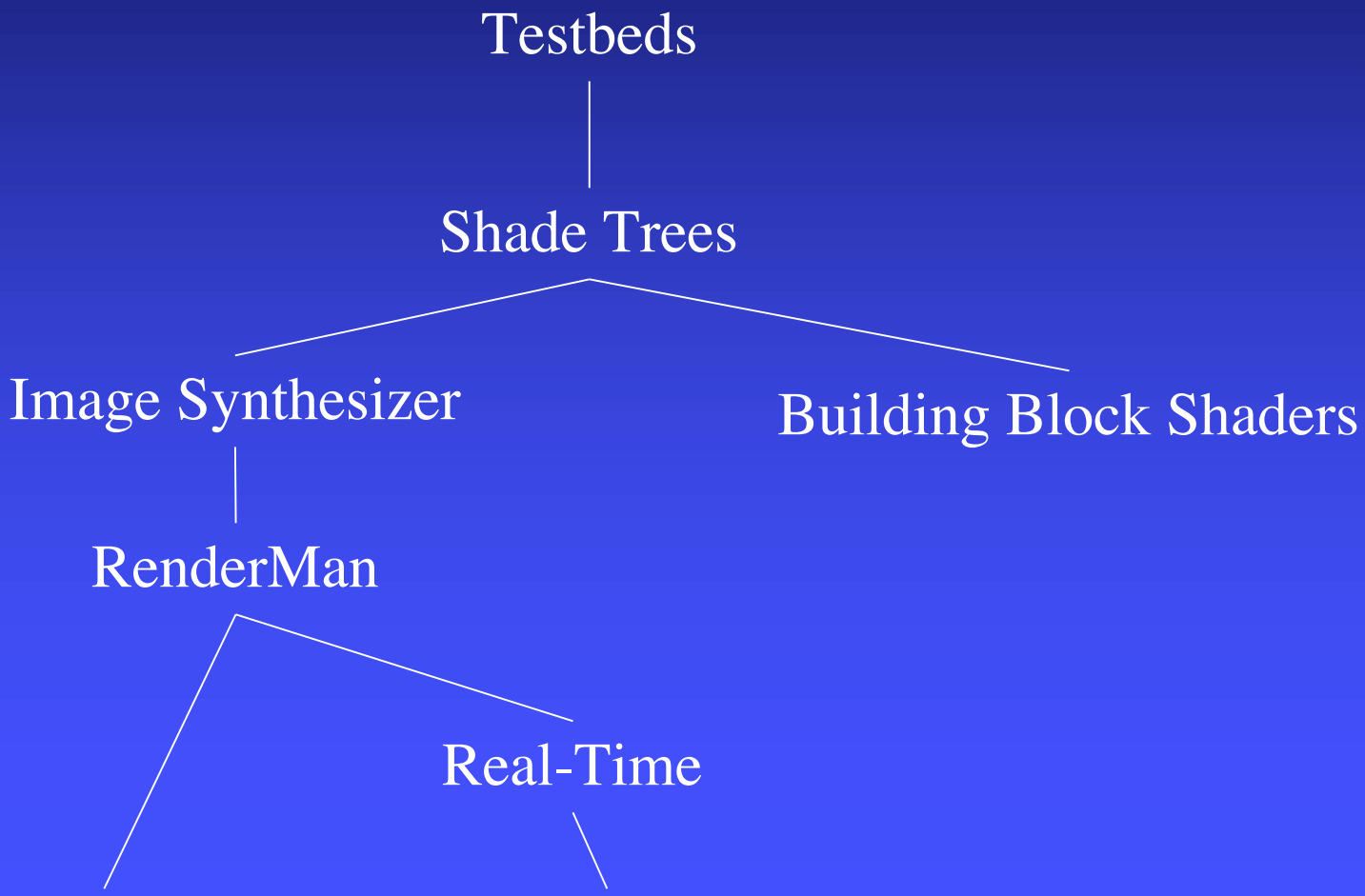
- Movies, commercials, etc.

[Toy Story – Pixar]

Shading View of the World



Evolution of Shading



Shader Point of View

- Written from point of view of one sample
- System decides
  - How many samples?
  - What sampling pattern?
- Same shader for many styles of rendering
  - REYES, Ray trace, SIMD

Special Purpose Language

- Example: RenderMan
  - Data types
  - Operators
  - Function types
  - Built-in functions
  - Language constructs

Data Types

- Frequency of execution
  - `uniform`: same at all samples
  - `varying`: potentially different
- Data
  - `string`
  - `float`
  - `point`, `vector`, `normal`, `color`
  - `matrix`

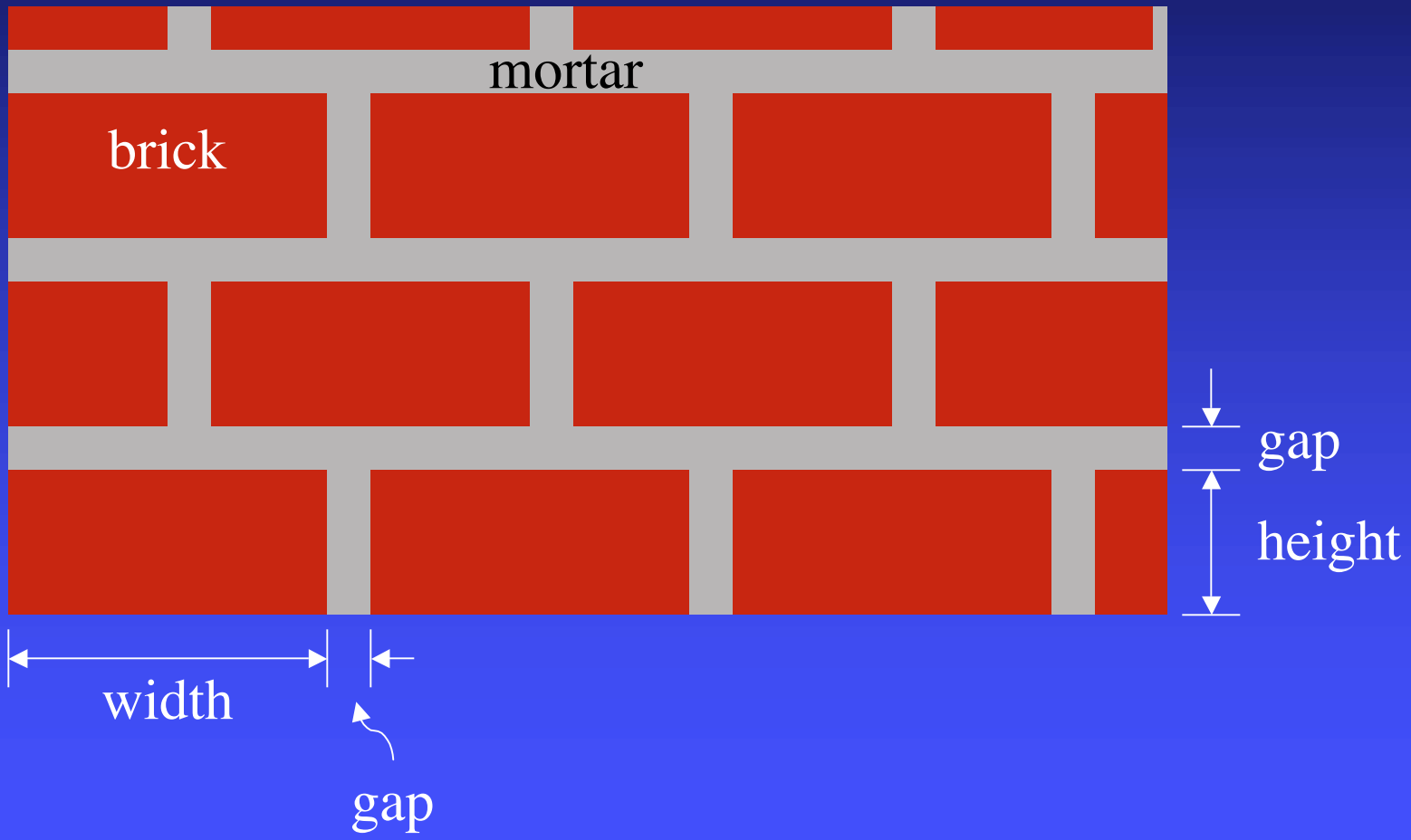
Function types

- Normal functions
  - Return any standard type
- Shading procedures
  - `surface`
  - `light`
  - `volume`
  - `displacement`
  - `imager`

Surface

- Use
  - Cs, Os
  - u, v, du, dv, s, t
  - P, N, Ng, dPdu, dPdv
  - E, I
  - L, Cl, Ol
  - time, dtime, dPdtime
- Set
  - Ci, Oi

Example: Brick

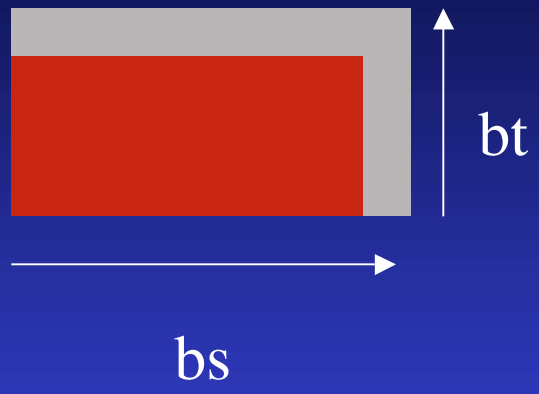


Brick Shader

```
surface brick(
        uniform float width = .2,
        uniform float height = .1,
        uniform float gap = .05,
        color brick = color(1,0,0),
        color mortar = color(.5,.5,.5)
)
{
  varying color bc;
  /* compute brick color */
  normal Nf = faceforward(normalize(N),I)
  Oi = Os;
  Ci = Oi*bc*(ambient()+diffuse(Nf));
}
```

Brick Color

- Where am I in my brick?
  - "brick coordinates"




```
varying float bs, bt;
/* compute brick coordinates */
if (bs < width && bt < height)
  bc = brick;
else
  bc = mortar;
```

Brick Coordinates

```
bt = mod(t, height+gap);
bs = s;
if (mod((t-bt)/(height+gap), 2) == 1)
  bs += (width+gap)/2;
bs = mod(bs, width+gap);
```

Variables Set by Shaders

- Displacement: P, N
- Surface: Ci, Oi
- Light: Cl, Ol, (L)
- Volume: Ci, Oi
- Imager: Ci, Oi

Operators

- Vector (point, normal, color)
  - Standard float ops work per-element
    - vector + vector, vector * vector
  - Dot product: vector . Vector
  - Cross product: vector ^ vector
- Standard matrix math
  - matrix + matrix, matrix - matrix
  - matrix * matrix, matrix / matrix
  - matrix * vector, vector * matrix

Built-In Math Functions

- The usual suspects (tan, atan, floor, …)
- `radians`, `degrees`
- `inversesqrt`
- `mod`
- `min, max, clamp`
- `mix`
- `step, smoothstep, filterstep`

Built-in Vector Functions

- Element access
  - `xcomp, ...; setxcomp, ...`
  - `comp, setcomp`
- `length, normalize`
- `distance, ptlined`
- `rotate, translate, scale`
- `Transform`

Built-in Matrix Functions

- `comp`, `setcomp`
- `rotate`, `translate`, `scale`
- `transform`
- `determinant`

Built-in Derivative Functions

- `Du`, `Dv`, `Deriv`
- `area`
- `calculatenormal`

Built-in Shading Functions

- `faceforward`
- `reflect`, `refract`, `fresnel`
- `trace`
- Lighting
  - `ambient`, `diffuse`, `specular`, `phong`
  - `specularbrdf`
- `texture`, `shadow`, `environment`

Other Built-in Functions

- `spline`
- `noise`
  - `pnoise`, `cellnoise`
- Communication
  - `surface`, `light`, ..

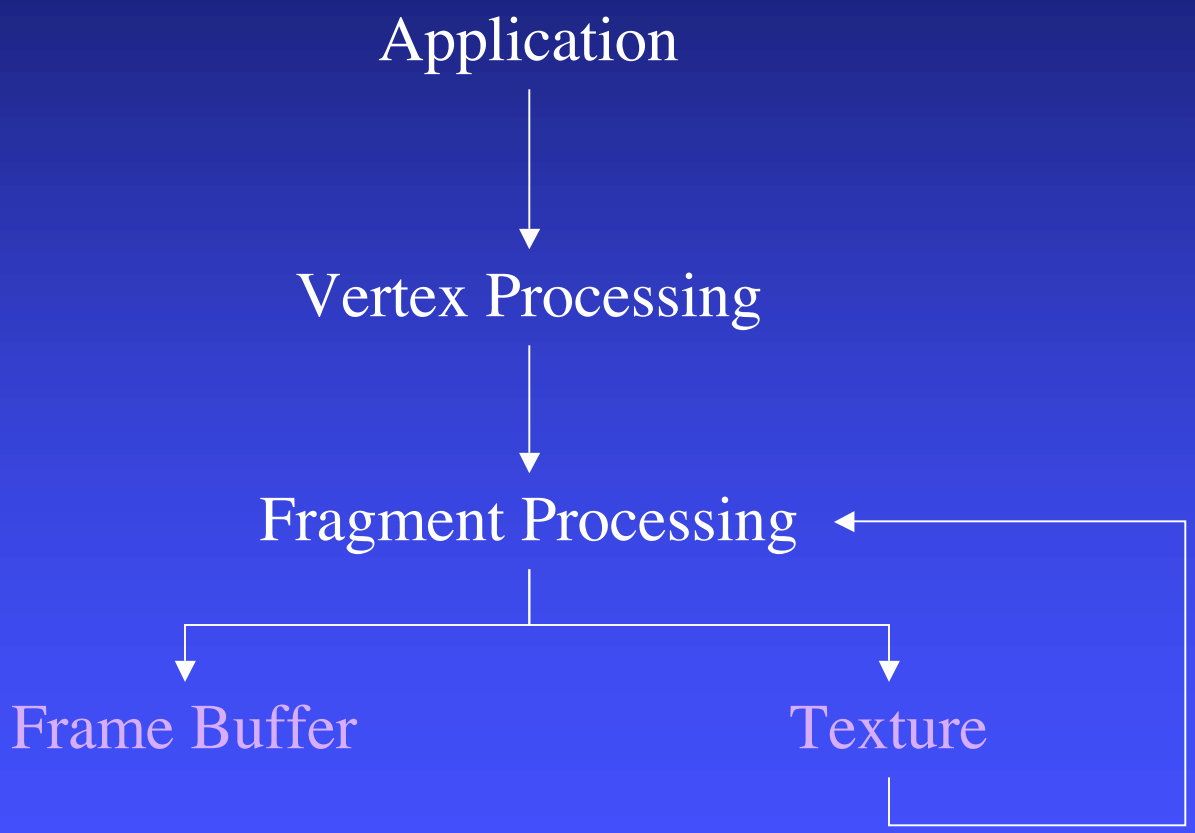
Language Constructs

- Accumulate light, surface shader
  - `illuminance(P,[N,angle]) { }`
  - Integrate/loop over lights hitting P
  - Use L, Cl, Ol inside
- Cast light, light shader
  - `illuminate(P,[axis,angle]) {}`
  - `solar([axis,angle]) {}`
  - Cast light from P
  - Set Cl, Ol inside; implicitly sets L

But I Want Real-Time!

- Graphics Hardware, the simple view
- Vertex processing
  - Transforms, per-vertex lighting, …
- Fragment Processing
  - Per-pixel lighting, texturing

Graphics Hardware



The Imperfect World of RTS

- Pixel-Planes 5
  - – low-level, – HW?
- Pixel-Flow
  - + RMan-like, – HW?
- SGI Multi-pass RMan
  - + RMan, – HW never built, – not available
- SGI OpenGL Shader
  - + any OpenGL HW, – low-level
- Stanford RTSL
  - + multi-platform, – no control flow
- Cg / DX9 HLSL
  - + high-level, + wide support thru DX, – v/f split, – no virtualization, – really a family of similar languages
- GLSL
  - + high-level, + requires virtualization, – v/f split, – targets future HW
- Ashli
  - + RMan SL, + handles virtualization, – partial coverage, – new HW only