GDC Tutorial:
Advanced OpenGL Game Development

A Practical and Robust Bump-mapping Technique for Today's GPUs

March 8, 2000

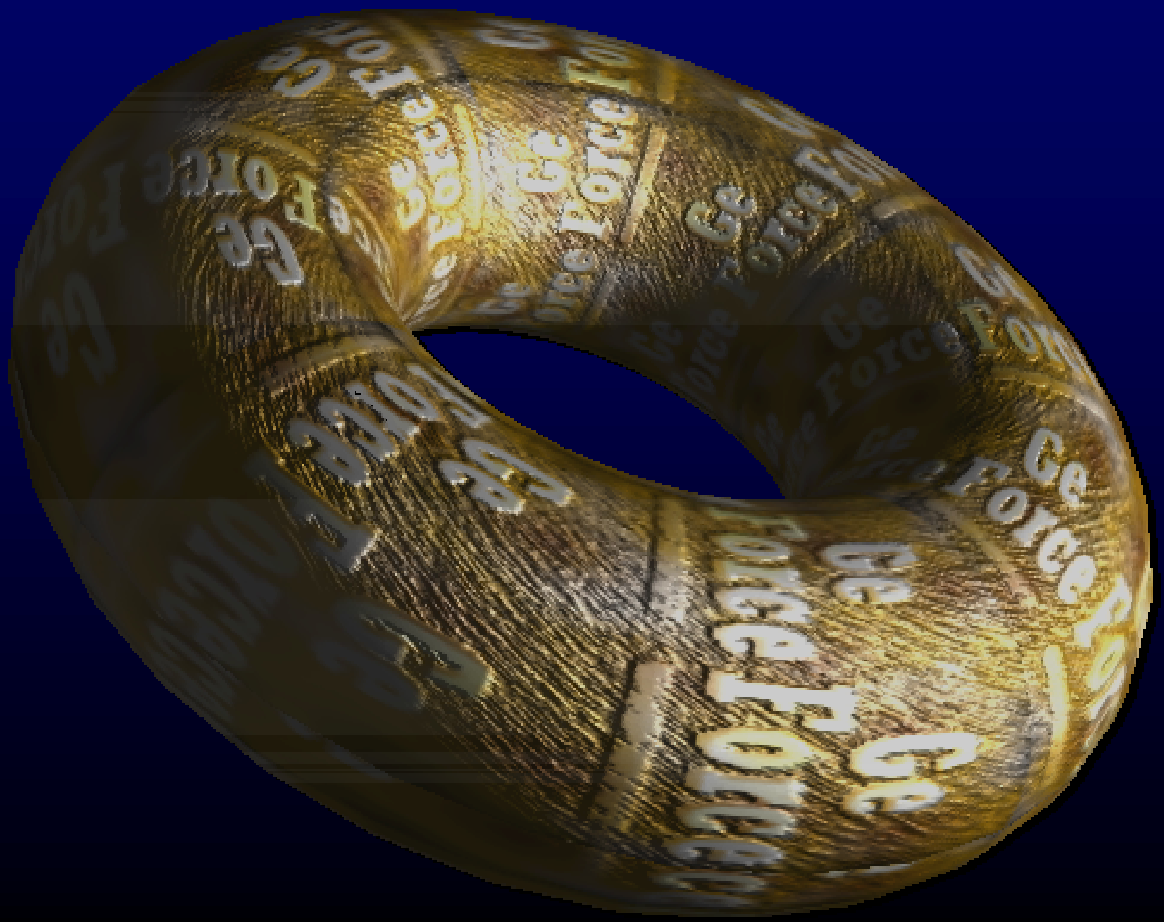
Mark J. Kilgard
Graphics Software Engineer
NVIDIA Corporation

“Hardware Bump Mapping” mostly hype so far

Previous techniques

- Prone to aliasing artifacts
- Do not correctly handle surface self-shadowing
- Cut too many corners, i.e. fail to renormalize, etc.
- Too expensive: many passes (OIM),
  needs accumulation buffer or glCopyPixels
- Only handle infinite, non-attenuated, non-spotlight lights
- Assume local viewer
- Correct only for essentially flat surfaces
- Difficult to implement and author

Demo!



Overview of the Technique

Uses new NVIDIA GPU features

- Cube maps - for per-pixel normalization
  - EXT_texture_cube_map extension
- Register combiners - for per-pixel dot products and additional math
  - NV_register_combiners
- And dual-textured - for normalization cube map and 2D normal map
  - ARB_multitexture

Features of the technique

Correctly handles

- Surface self-shadowing for both diffuse and specular contributions
- Diffuse minification so that bumped objects in the distance look dimmer than equivalent smooth distant objects
- Mipmap filtering of normal maps minimizes aliasing
- Local, attenuated, and/or spot-light light sources with local viewer
- Ambient and surface decals

Anatomy of the technique

GAMEDevelopers CONFERENCE 2000

Three passes (or just two if no specular)

diffuse × *decal* + *specular*

High-level view

Tangent space per-pixel lighting [Peercy 97]

- CPU supplies tangent space light vector or half-angle vector as (s,t,r) texture coordinates
  - linearly interpolated
  - normalized by an RGB8 cube map texture
- Normal map is RGBA8 2D texture
  - RGB contains “compressed” normalized perturbed normal
  - Alpha contains averaged normal shortening
- But no hard-wired per-pixel “lighting engine”!

Per-pixel lighting building blocks

Cube maps enable per-pixel vector normalization

- Cube map normalizes interpolated (s,t,r) into RGB
  - (s,t,r) is interpolated per-vertex light or half-angle vector
- Combiners expand RGB from [0,1] range to [-1,1] range
  - expansion done by register combiners
- *Optimization*: If infinite viewer or infinite light used, half-angle can be computed within the cube map!
- 32x32x6 RGB8 cube map texture is sufficient

Per-pixel lighting building blocks

Register combiners perform per-pixel math

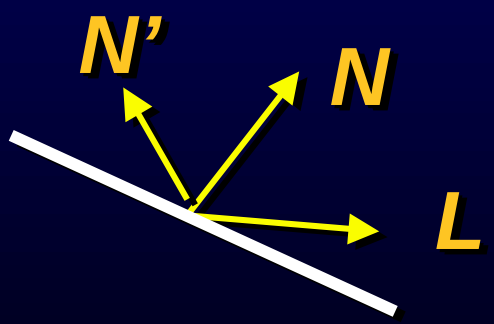
- Signed math for dot products
  - L dot N' for diffuse, H dot N' for Blinn specular
  - L and H properly normalized by cube map
  - Pre-normalized N' supplied by normal map
- Uses tangent space L_z for geometric surface self-shadowing
- Constant color for adding in ambient contribution
- Successive squaring for specular exponent

Surface self-shadowing

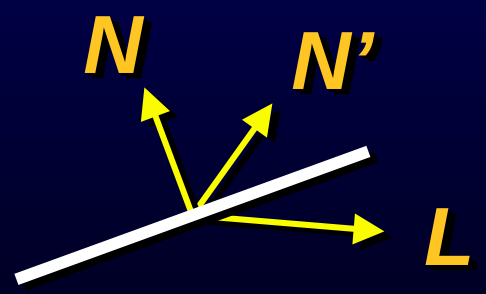
GAMEDevelopers CONFERENCE 2000

Two kinds of self-shadowing

- max(0, L dot N') based on the perturbed normal
- Also should clamp when L dot N goes negative!



Surface <u>should self-shadow</u> due to perturbed normal, I.e., L dot N'<0



Surface <u>should self-shadow</u> due to unperturbed normal, I.e., L dot N<0

Surface self-shadowing

Self-shadowing computation

- Technique supports bumped self-shadowing by computing
  min(8*max(0,Lz), 1) * max(0, L dot N')
- Also stashes this self-shadowing term in destination alpha
  min(8*max(0,Lz), 1)
- Specular pass blends with DST_ALPHA,ONE computing
  min(8*max(0,Lz), 1) * max(0, H dot N')^8
- Illumination does not appear on geometric "back side"
  - Steep ramp avoids winking and popping at self-shadow boundary

Complete lighting model

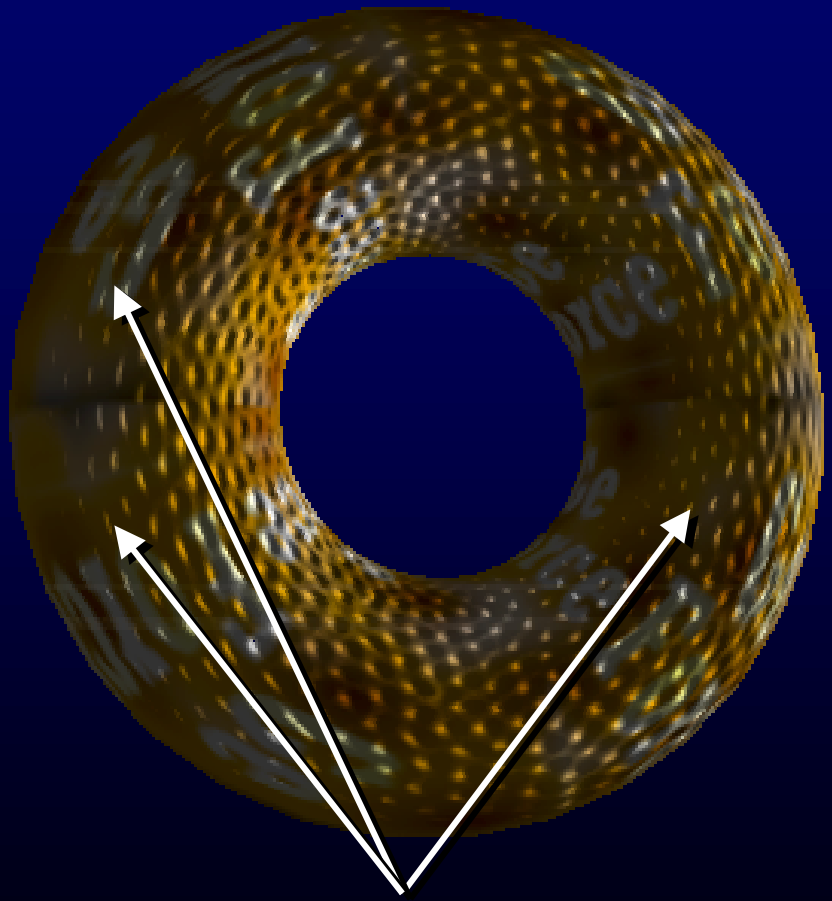
More per-pixel math than you might think!

- Ambient = constant
- Diffuse = min(8*max(0,Lz), 1) * max(0, L dot N')
- Specular = min(8*max(0,Lz), 1) * max(0, H dot N')^8
- Final = (Diffuse + Ambient) * Decal + Specular

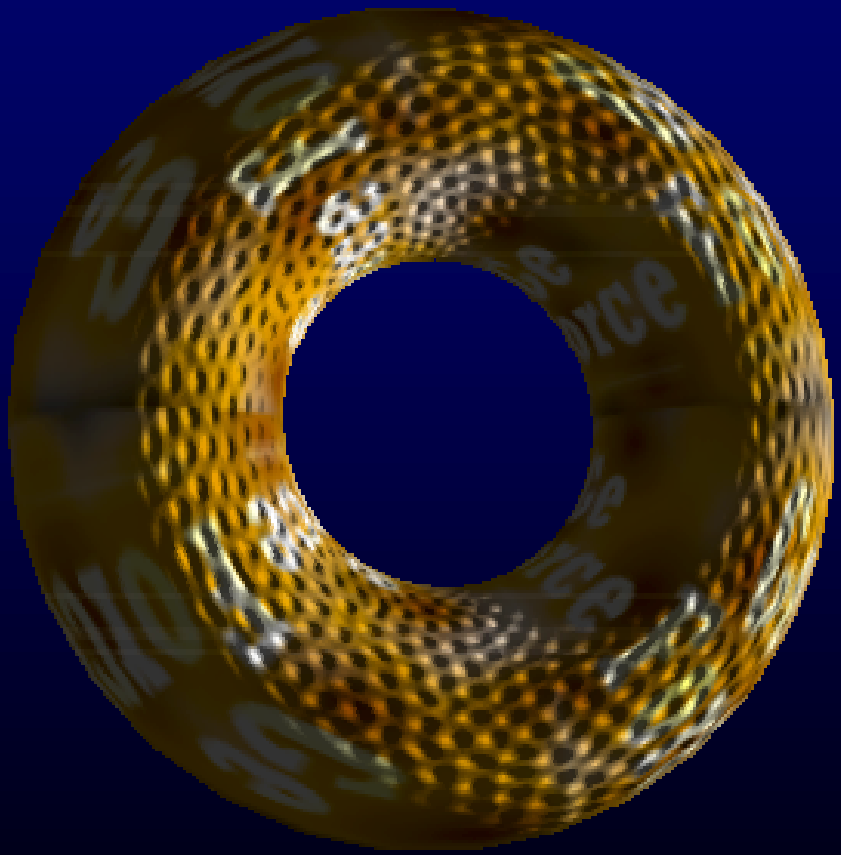
Attenuation and/or spotlight

- Final = (Attenuate*Diffuse + Ambient) * Decal + Attenuate*Specular

Self-shadowing example



Light leaks onto torus “back side” due to bad geometric self-shadowing



Proper geometric self-shadowing!

Some CPU work required

Per-vertex work

- Model requires per-vertex tangent space basis
  - any bump mapping scheme needs surface normals (N), tangents (T), and binormals (B)
- If light or viewer (in the specular case) changes relative to the object, CPU re-compute tangent space light and half-angle
  - Transform object-space vector by [T B N] 3x3 matrix
- *Nice split:* CPU does per-vertex work, GPU does transform and per-pixel work

Issues

Truth in advertising

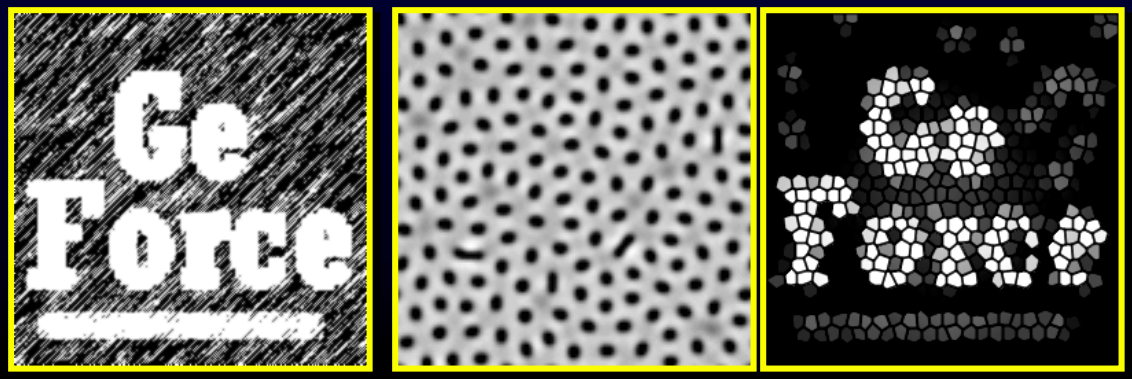
- Specular exponent only 8 so not too shiny
  - banding evident in specular highlights, though easy to hide artifacts in surface decal and diffuse contribution
- Perturbed normal slightly de-normalized by filtering
  - causes very slight dimming, alternative is aliasing
  - most objectionable when magnifying, “grated” look
- Not “bumped environment mapping”
  - but DX6 bump-env support is a joke
- Requires GeForce 256, Quadro, or any future NVIDIA GPU

Claim: practical

GAMEDevelopers CONFERENCE 2000

Justification for this claim

- Real-time frame rates on current generation GPUs
  - tweakable: eliminate specular or decal pass, etc.
- Mixable with other multi-pass rendering techniques
  - stenciled shadows, multiple lights, fog, etc.
- Bump map easy to author as an 8-bit gray-scale image
  - examples:



Claim: robust

Justification for this claim

- Animates with minimal temporal aliasing
- Models both self-shadowing terms, including for specular
- Diffuse illumination filtered reasonably
- Light and half-angle vectors renormalized per-pixel
- Local, attenuated, and spot-light light sources
- Local viewer
- Overall, very reasonable fidelity to Blinn’s original formulation

Source available on the web already

Bumpdemo source code

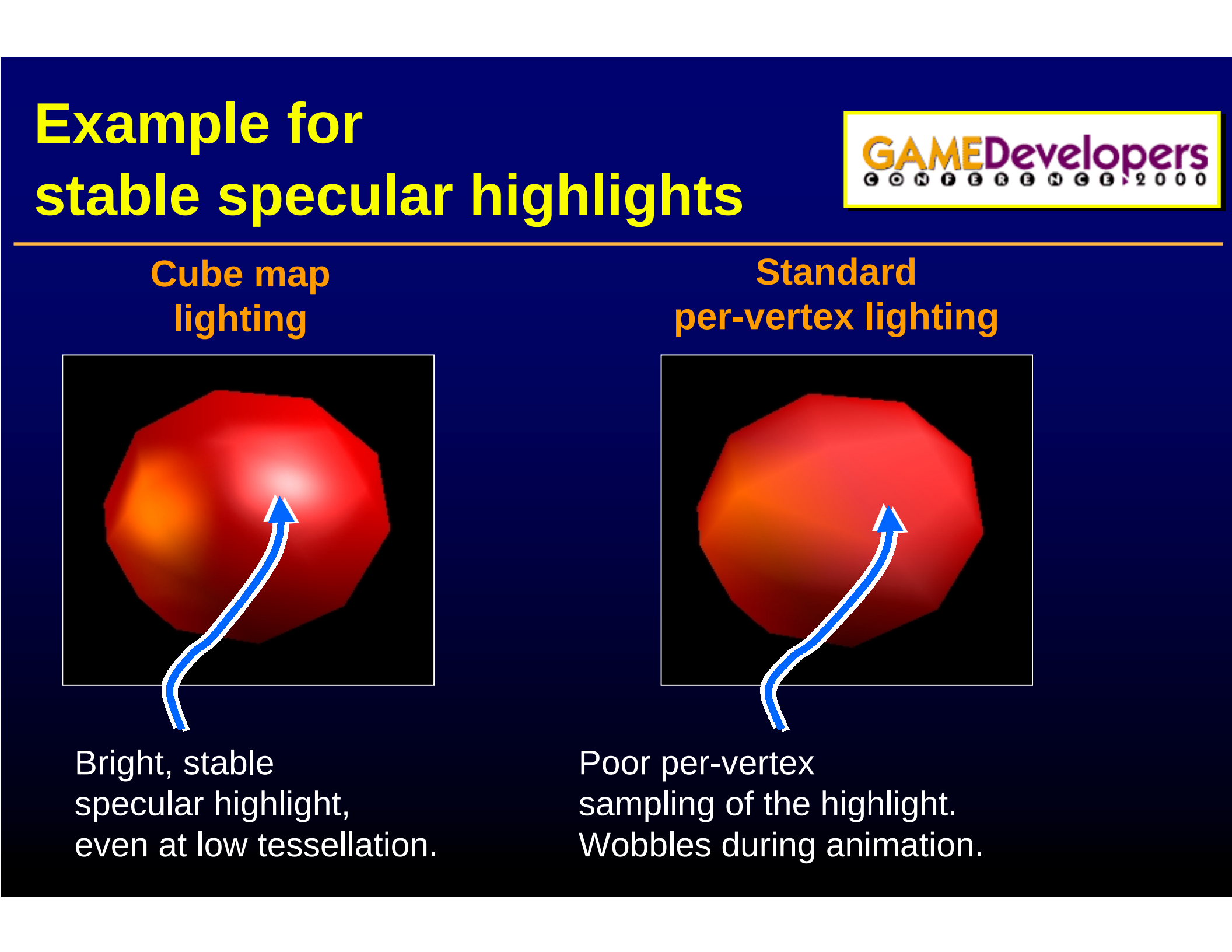
- Go to www.nvidia.com Developer Section
  - http://www.nvidia.com/developer.nsf/htmlmedia/devMainFR_top.html
- Find and download bumpdemo.zip
  - requires GeForce 256 or Quadro to run
- Also download “NVIDIA OpenGL Extension Specifications”
  - nvOpenGLspecs.pdf
  - Adobe Acrobat, 197 pages
  - documents OpenGL extensions used by bumpdemo

Other novel uses for cube mapping

Stable specular highlights

- Encode specular & diffuse lighting solution
  - Diffuse cube map using *normal* map texgen
  - Specular cube map using *reflection* map texgen
  - Encode unlimited number of directional lights
- Result is stable specular highlights
- Less significant for diffuse lighting
  - Average of dot products ≅ dot product of averages
  - Excepting clamping

Example for stable specular highlights



Cube map lighting

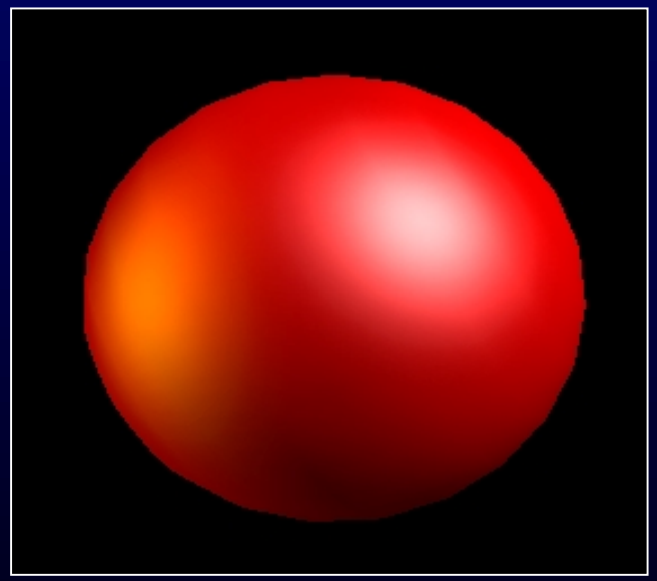
Bright, stable specular highlight, even at low tessellation.

Standard per-vertex lighting

Poor per-vertex sampling of the highlight. Wobbles during animation.

Another example of stable specular highlights

High tessellation does not completely fix per-vertex artifacts



Still bright and stable.

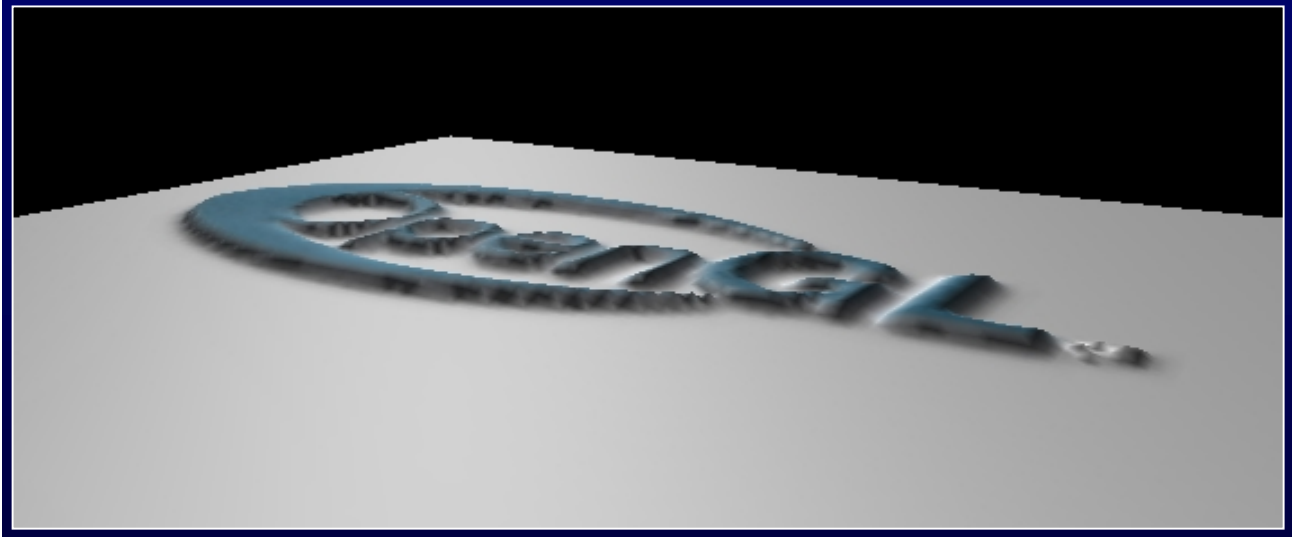
Better sampled, but still has streaky artifacts.

Still more novel uses for cube maps & combiners

Per-pixel specular normal mapping

- Assumes fancy per-pixel dot product operations!
- Cube map encodes normalized half-angle
  - Texgen supplies view vector
  - Cube map generates normalize(V+L) vector
- Another 2D texture supplies per-pixel surface normals
- Per-pixel specular lighting
- Per-pixel “normalized(V+L) dot N” !

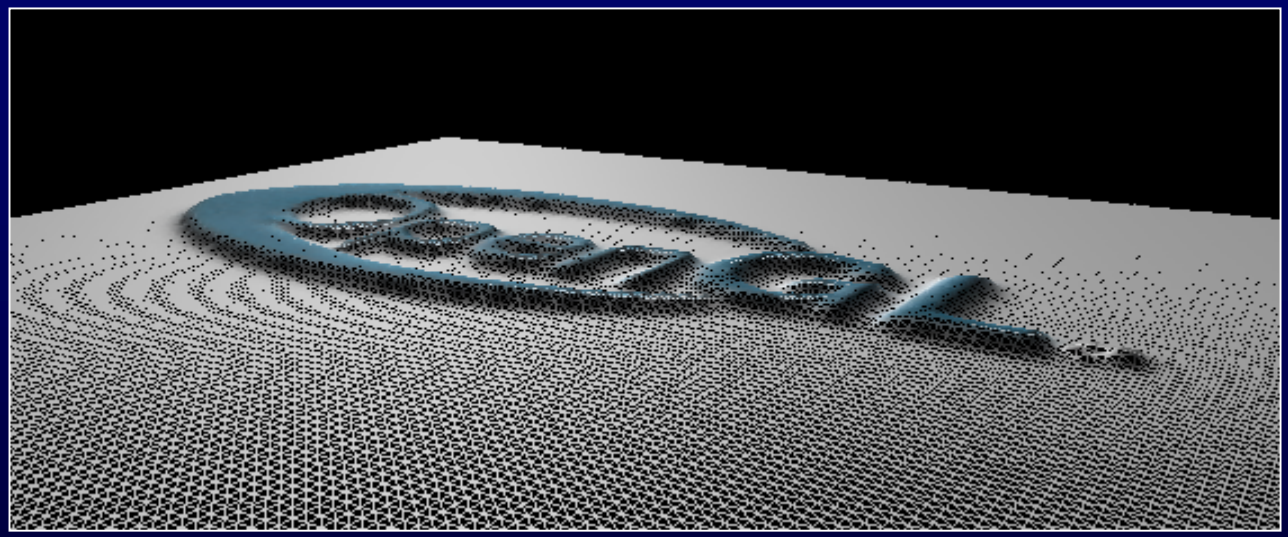
One pass per-pixel diffuse & specular example!



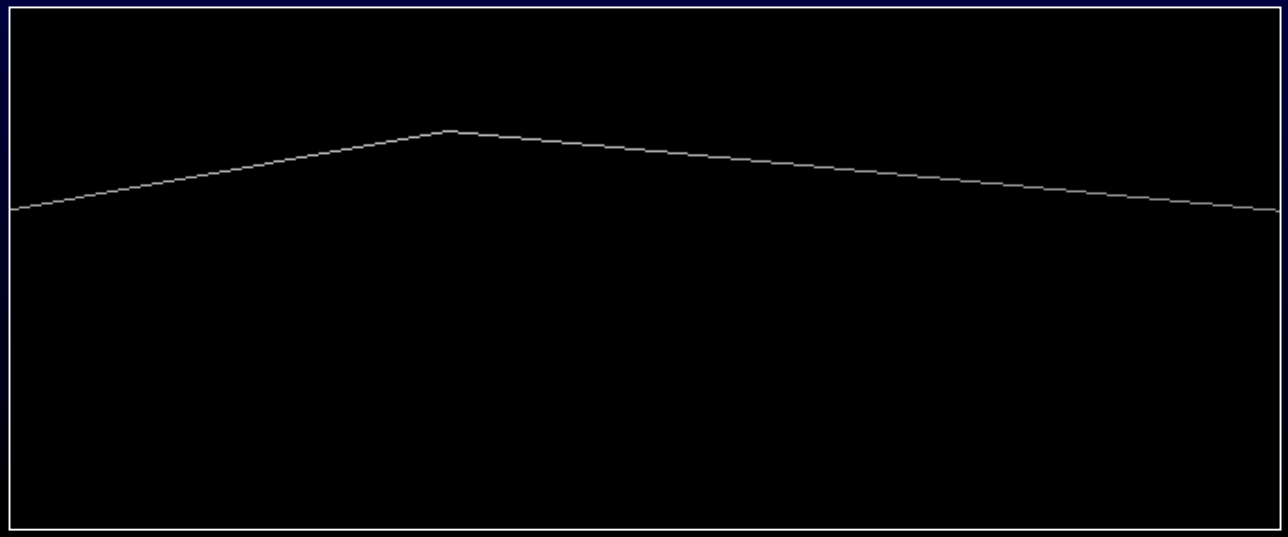
OpenGL per-vertex lighting with mesh of thousands of vertices (note displacement).

Per-pixel specular normal mapping with cube maps (flat, but lighting all there).

Example in wireframe

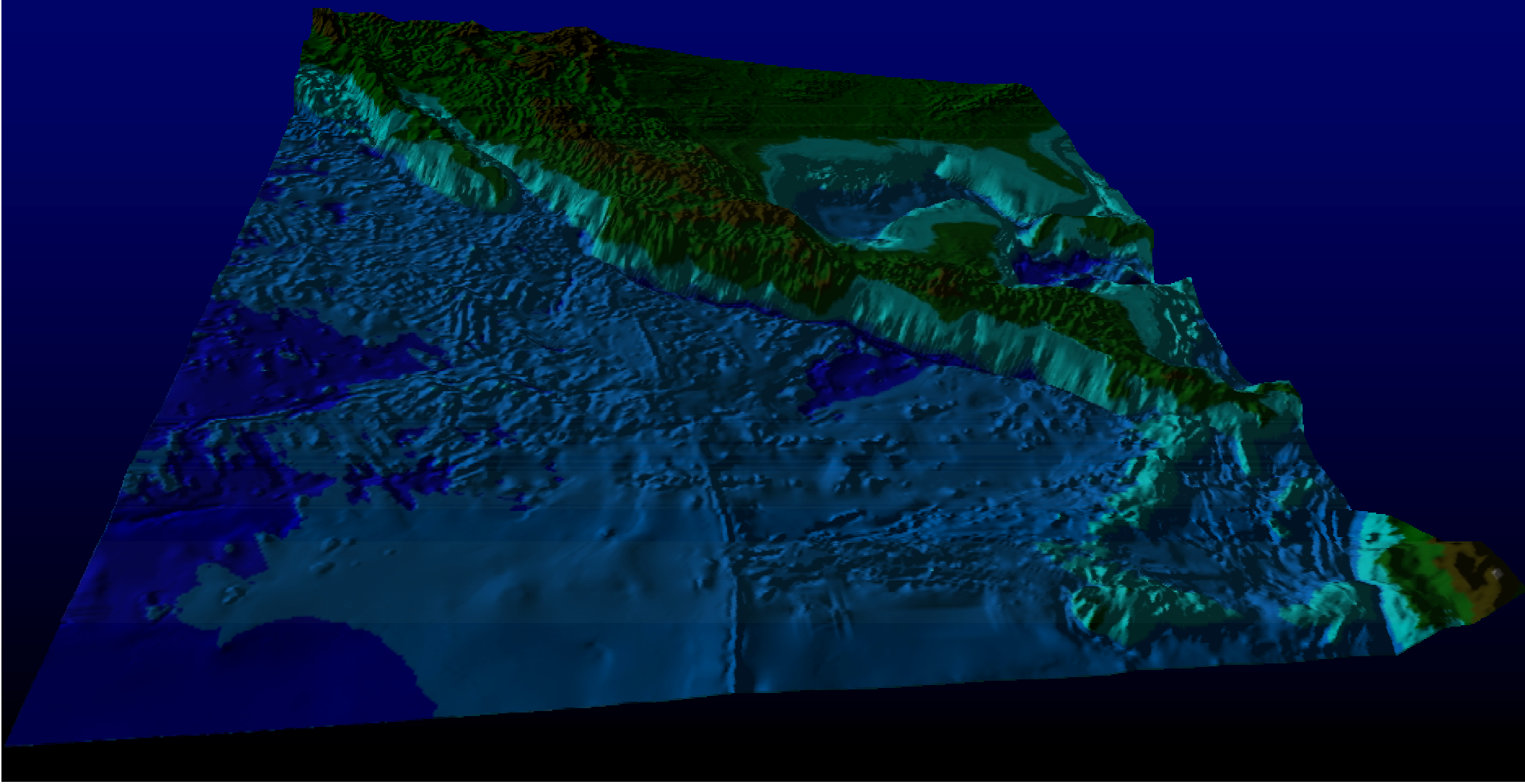


Dense, dense mesh.

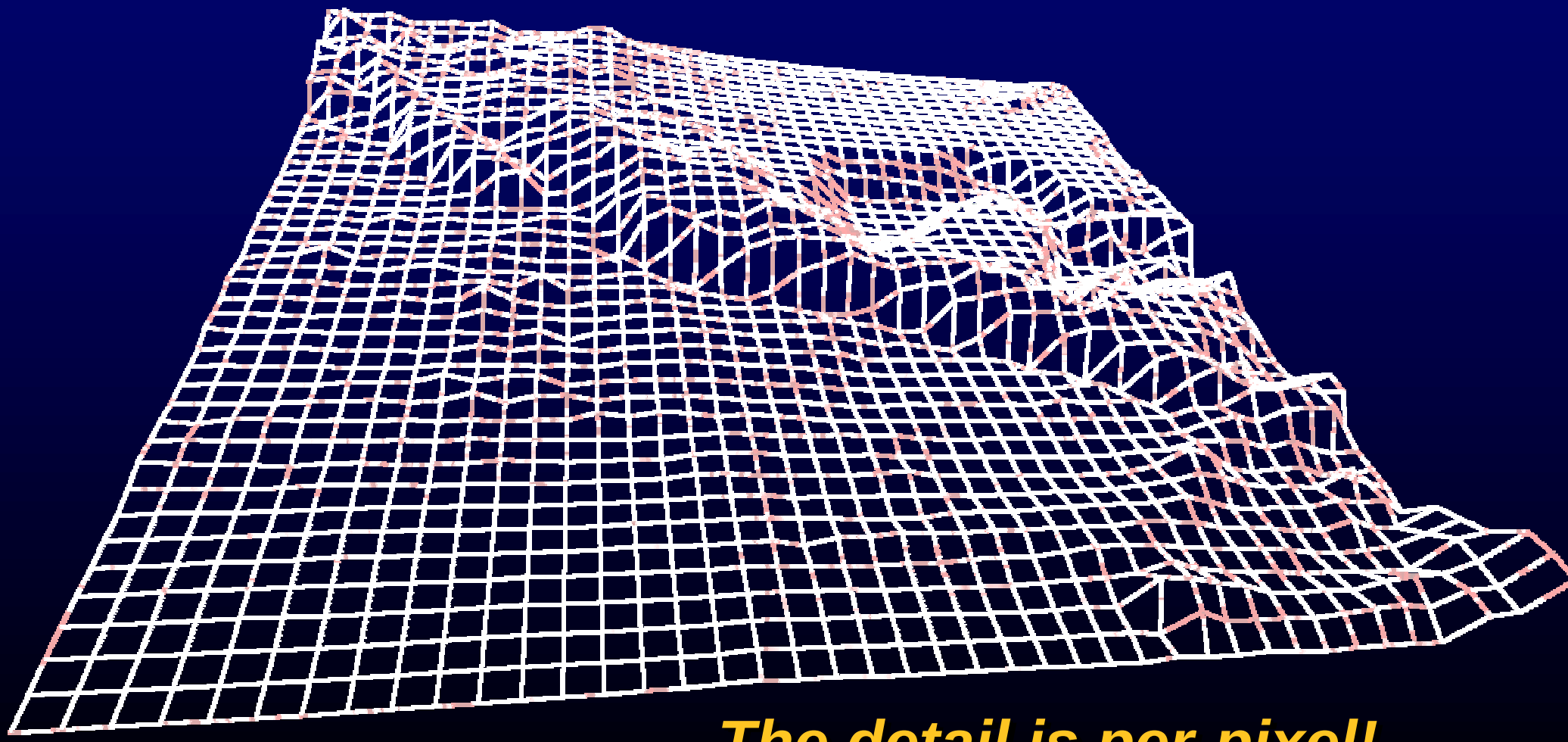


One polygon!

Object-space per-pixel lighting example



Object-space per-pixel example in wire-frame

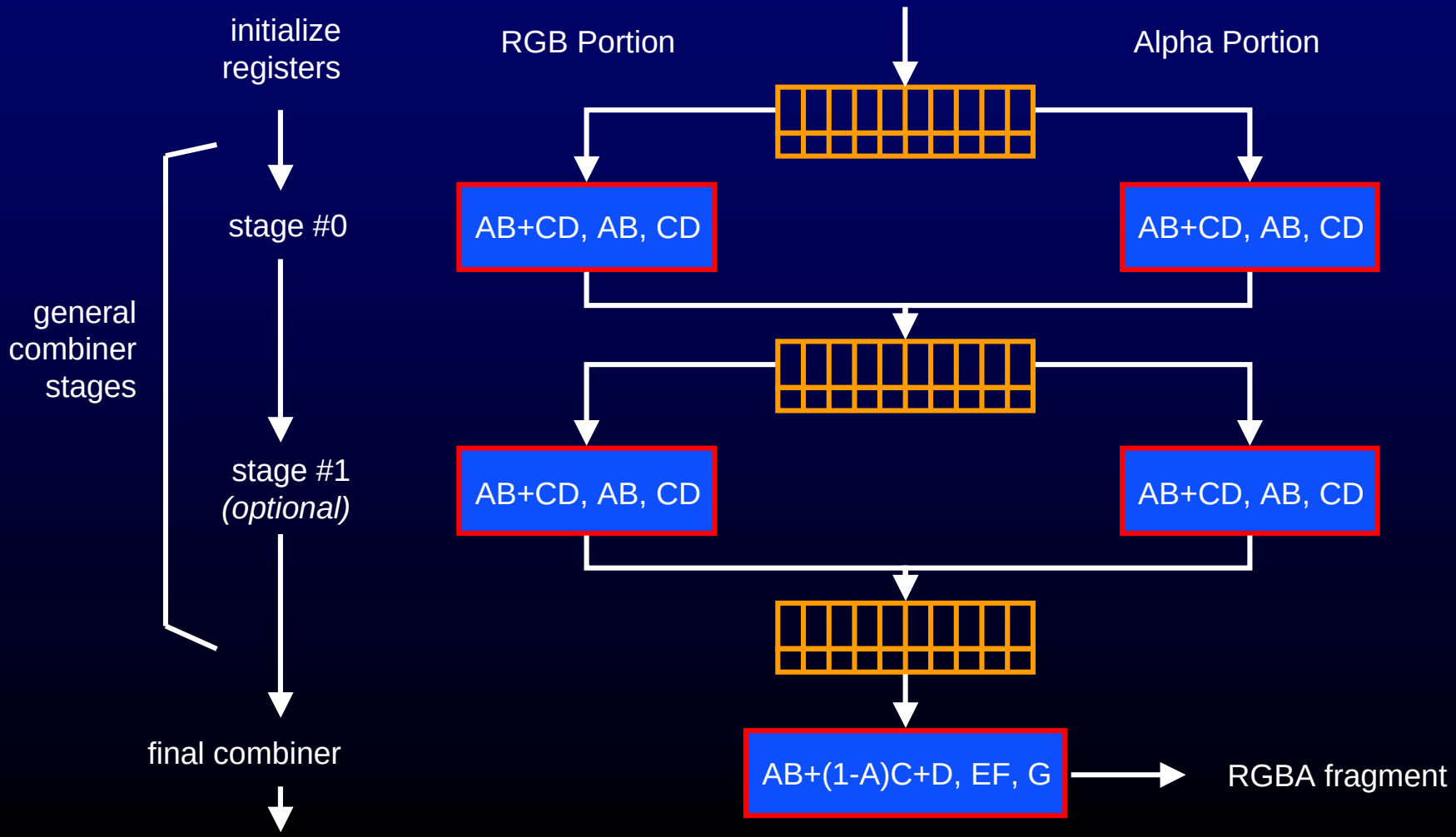


The detail is per-pixel!

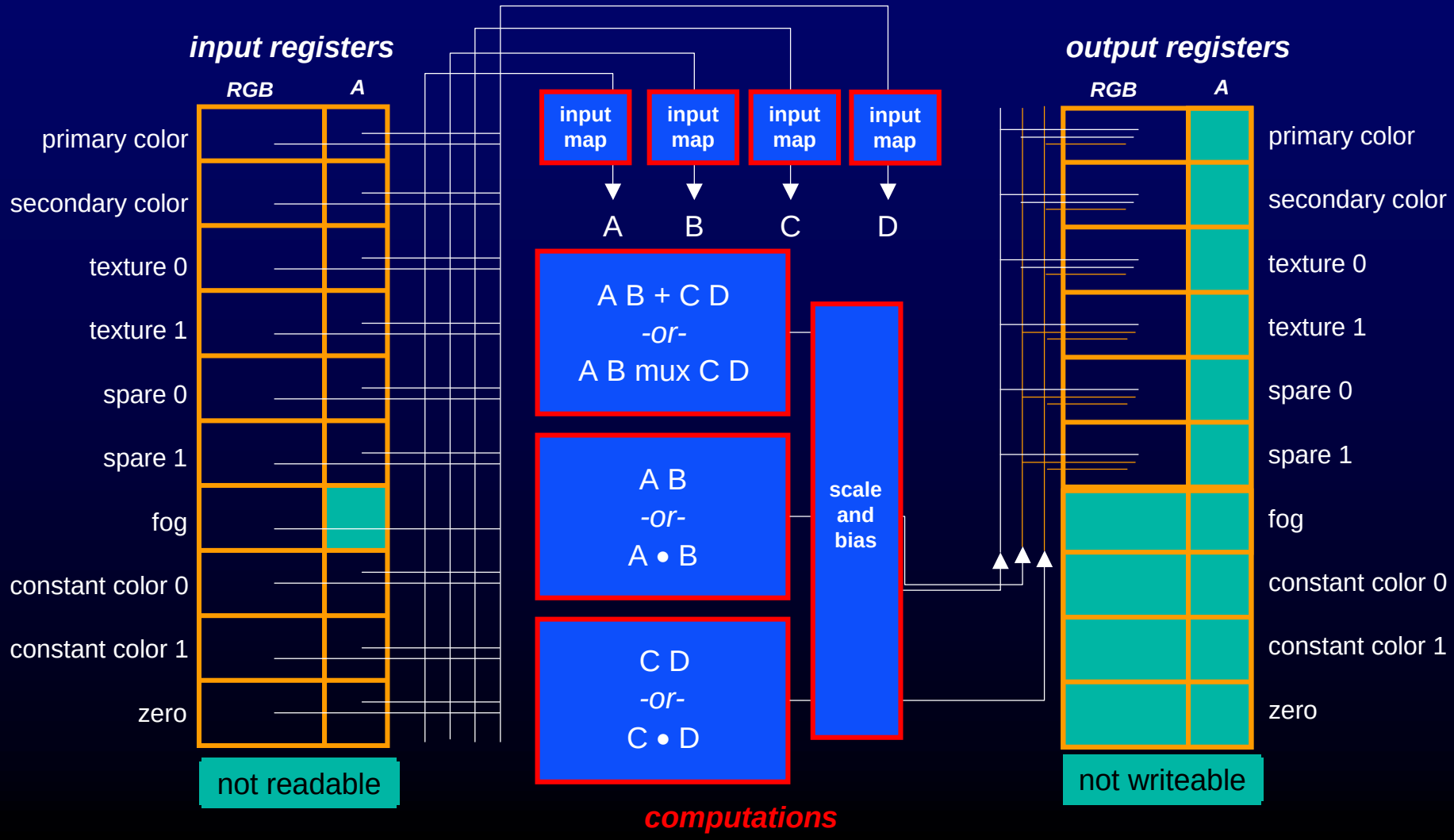
Register combiners overview

- overrides texture stages/environment, color sum, and fog in current APIs
- signed math (negative one to positive range)
  - extended range through scaling
- dot products for lighting and image processing applications
  - designed for specular, diffuse, and ambient per-pixel lighting
  - object space bump map lighting
  - tangent space bump map lighting
  - post-filtering 3x3 color matrix for color space conversions
- register model supports non-linear data flows
  - superior to linear chain in current APIs
- effectively, a VLIW instruction set for fragment coloring
- very efficient hardware implementation

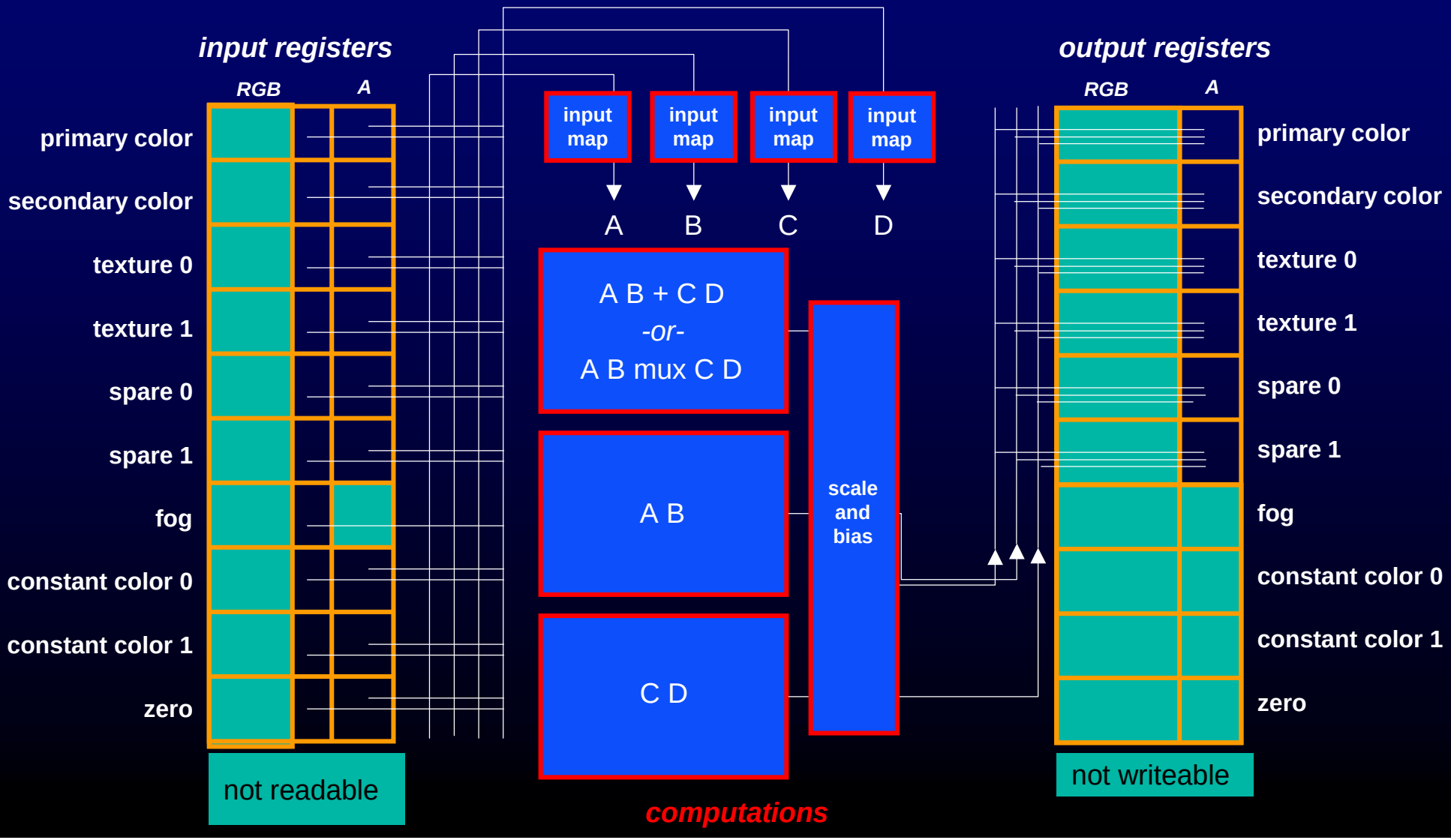
Register combiners operational overview



General combiners
RGB operation



General combiners
Alpha operation



Final combiner operation