Stanford Shading Group presentation

Hardware Bump-mapping Choices and Concepts

May 8, 2000

Mark J. Kilgard
Graphics Software Engineer
NVIDIA Corporation

Coordinate spaces for lighting

Lighting math is rotationally invariant

- Pick the most efficient coordinate space for lighting
- Possible choices
  - eye space (OpenGL per-vertex lighting)
  - world space
  - object space
  - tangent space
- Not screen space since that involves a projection

Evaluating Lighting Coordinate Systems

Eye space

- Advantages
  - eye-space position = view vector
  - nice if hardware floating point dot products are cheap
- Disadvantages
  - lights must be repositioned per-view
  - object-space coordinates must be transformed

Evaluating Lighting Coordinate Systems

World space

- Advantages
  - stationary lights do not need to be repositioned per-view
  - world-space cube maps can encode distant specular illumination
- Disadvantages
  - view vectors harder to compute
  - everything has to be transformed into world space

Evaluating Lighting Coordinate Systems

Object space

- Advantages
  - object space normals need no transformation
- Disadvantages
  - lights and eye must be transformed into object-space
  - view vectors harder to compute

Evaluating Lighting Coordinate Systems

Tangent space

- Advantages
  - normal is always straight up, i.e. (0,0,1)
  - makes it easy to perturb for bump mapping!
  - tangent readily available for anisotropic lighting
- Disadvantages
  - lights and eye must be transformed into object-space
  - view vectors harder to compute

Coordinate Systems for Bump Mapping

Common choices

- tangent space
- object space
- but any could work
  - but we want an “easy” space for bump mapping

Tangent Space Bump Mapping

Easy to perturb the normal

- Last column of perturbation rotation matrix “is” the perturbed normal
- By making assumptions, bump map can be “de-coupled” from the object geometry
  - square patch assumption [Peercy 97]
  - saves texture memory since bump maps can be used on different geometry
  - acceptable for morphing geometry

Object Space Bump Mapping

Directly encodes object-space normals

- encode normals directly in texture map
  - but bump map texture is tied to object geometry
  - morphing objects are difficult
  - no need for square patch assumption (though still often assumed)

Encoding Bump Maps

Common choices

- normal maps
  - encode direction vectors (in whatever space)
  - no re-normalization required
- offset maps
  - encode orthogonal offsets from unperturbed normal
  - requires re-normalization
- height fields
  - must approximate derivatives during rendering

Normal maps

Encodes normal vectors directly

- filtering may require re-normalization
  - but not as important
- just perform per-fragment lighting directly using this normal
- diffuse filtering is tractable
- arbitrary normals possible
- scaling issues

Offset maps

Bias unperturbed vector and re-normalize

- cannot encode arbitrary normals
- just Dx, Dy need encoding
  - but large range is needed and filtering is an issue
- requires re-normalization
  - often coupled with cube maps for re-normalization
  - supports so-called “bump environment mapping”
  - but this requires rotations into cube-map space

Height fields

Encodes height of displacement directly

- requires on-the-fly approximation of derivatives
  - usually done with finite differencing or “embossing”
- embossing works poorly when light is straight-on
- very prone to filtering artifacts

Per-fragment Lighting Model

What per-fragment computation

- Usually conventional diffuse/specular/ambient model
- Lambertian diffuse (L • N)
  - straightforward
- Specular choices
  - Phong specular (R • L)
  - Blinn specular (H • N)

Phong Specular Bump Mapping

Often used with offset maps

- Compute un-normalized reflection vector
  - [Voorhies 94]
- Encode specular solution in a cube map
  - rotate perturbed reflection vector into world space
  - cube map provides normalization and exponentiation
  - so-called “free” bump environment mapping
  - do not ignore cost of generating and storing cube map!

Blinn Specular Bump Mapping

Direct approach

- Originally described for bump mapping [Blinn 78]
- Use perturbed normal directly
  – no reflection vector computation
- Requires per-fragment signed dot products
- Evaluating the exponential
  – successive squaring or a look-up table
  – both lead to banding for fixed-point math

Lighting Computation Efficiency

Two approaches

- direct per-fragment lighting computation
  - GPU bump mapping [Kilgard 00]
- pre-computing lighting
  - quantized normal maps [Tarini et.al. 00, Miller et.al. 98]
  - cube maps [Ernst et.al. 95]
  - paletted textures