Adding Spherical Harmonic Lighting to the Sushi Engine

Chris Oat

3D Application Research Group

ATI Research, Inc.

Overview

- Introduction & Motivation
  - Quick Review of PRT
- Case Study : ATI’s demo engine “Sushi”
  - Design Goals
  - Our Workflow
  - Implementation
- Demo
- Working around PRT limitations
  - Using Blockers & Receivers
  - Lighting Considerations
  - Animating
  - Reducing Memory Costs
- Conclusion

Motivation

- Share the lessons learned from our implementation
- Demonstrate the advantages of a shader-driven approach to SH and PRT
- How we worked around the various limitations of the technique

Global Illumination

- Non-local lighting
  - Area light sources
  - Shadows
  - Inter-reflections
  - Subsurface scattering
- Raytracing, Radiosity, etc.
- These are not real-time friendly

Rendering Equation

$$I_p = \int_S L_i(s) V_p(s) H_{N_p}(s) ds$$

Reflected Light Intensity	=	Incoming Light Intensity	*	Light Source Visibility	*	Hemisphere Cosine Term

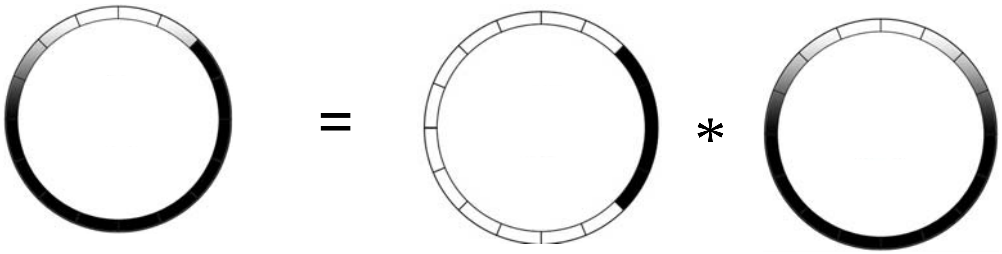
Rendering Equation Revisited

$$I_p = \int_S L_i(s) V_p(s) H_{N_p}(s)\, ds$$

- Constrain *V* and *H* so that they are constant
  - Model is rigid
  - Model does not move relative to it’s visible surroundings
  - Incoming light is on distant sphere
- Pre-compute these terms (*Pre-Computed Radiance Transfer*) and store at all points p

Transfer Function

$$T_p(s) = V_p(s) H_{N_p}(s)$$



- Transfer function encodes how much light is visible at a point and how much of that visible light gets reflected
- Store using spherical harmonic basis functions
- Integrating with incoming light is now just a dot product of two vectors

Real Time Global Illumination

- Preprocessor computes diffuse radiance transfer and stores this data per-vertex or per-texel
- Run-time engine projects lights into spherical harmonics
- Pixel/Vertex shader integrates incoming light with diffuse transfer for global diffuse reflection

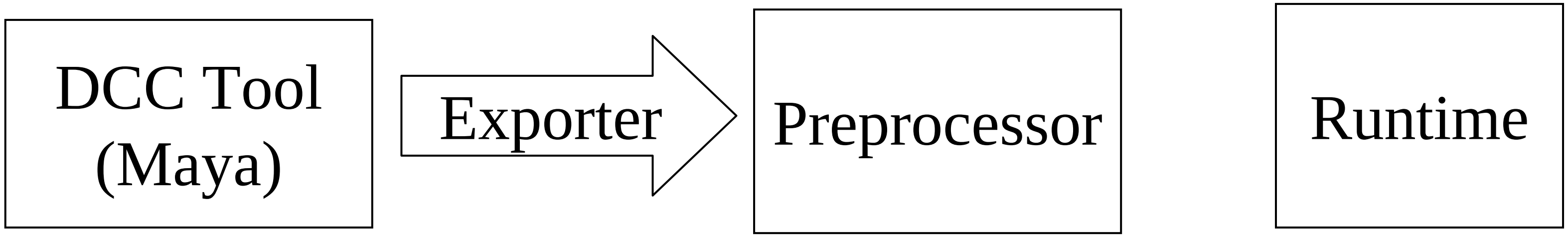
For a Full Review…

- Peter-Pike Sloan et al, SIGGRAPH 2003-2004
- Robin Green, “Spherical Harmonic Lighting: The Gritty Details”
- Tom Forsyth, “Spherical Harmonics in Actual Games”
- DirectX 9.0 SDK (Beta2 or higher)

Preprocessor Design Goals

- Existing workflow should not be interrupted
  - Modeling/Scene Setup : Maya
  - Export & Preprocess : Sushi Object Preprocessor
  - Runtime : Sushi Runtime
- Completely Shader Based
- Two Pass
  - Generate PRT data
  - Everything else
- Multiple PRT “Materials” per shader
- Light grouping

Workflow



DCC Tool (Maya)

Exporter

Preprocessor

Runtime

- Create Geometry
- Apply Shader
- Export Raw Data

- Read shader headers
- PRT Simulation
- Convert raw data to runtime format

- Load runtime data
- Load shaders
- Render

Shader Based Approach

- PRT enabled/configured in Shader’s header
- Allows shader author to configure simulator and define data targets for simulation results
  - Texture targets
  - Vertex buffer targets
  - Constant store targets

Shader’s Header Block

- Enables PRT type
  - Receiver
  - Blocker
- Configures Preprocessor/Simulator
  - Where the simulator gets its input from
  - Complexity of simulation (rays, bounces, etc)
  - Material properties
- Defines how the run time should pass coefficients to the shader
  - Per-Vertex
  - Per-Texel

Defining Receivers & Blockers

- Receivers
  - Have PRT Coefficients computed in during preprocess
  - Coefficients stored Per-Vertex or Per-Texel
  - Draw at runtime
- Blockers
  - Don’t get PRT Coefficients
  - Cast shadows onto Receivers
  - Do not draw at runtime

PRT Type

```
Prt[TYPE][INDEX]
```

- Type
  - **`Blocker`**
  - **`Receiver`**
- Index
  - Used to reference this block in other parts of the shader
    - Setting vertex buffer targets
    - Setting constant store targets
  - Allows multiple PRT simulations may be enabled in a given shader
- Technically, no further settings are required but the defaults aren’t very interesting…

Configuring a PRT Receiver

```
PrtReceiver0 SHOrder(6) Rays(2048) Bounces(3) SSS(1) Spectral(1)
```

- **`SHOrder()`** : Order of Spherical Harmonic Approximation (n^2 Coefficients)
- **`Rays()`** : Rays fired per-sample (vertex/texel)
- **`Bounces()`** : Number of bounced light interations
- **`SSS()`** : Enabled/Disable Subsurface Scattering Simulation
- **`Spectral()`** : Enable/Disable Spectral

Target Settings

```
PrtReceiver0 ... HighQualityCPCA(1) PCAWeightVectors(7) PCAClusters(1) TexTargets(“tPCA0”,…)
```

- **`HighQaulityCPCA()`** : Enable/Disable high quality CPCA compression of transfer coefficients
- **`PCAWeightVectors()`** : Number of PCA Weight Vectors (n*4 = number of PCA Weights)
- **`PCAClusters()`** : Number of PCA Clusters
- **`Dilation()`** : For texture targets, dilate results to remove texture filtering artifacts
- **`TextureTargets()`** : list of textures that get filled with PCAWeights, one 4 channel texture is needed per-PCA Weight vec
  - If target isn’t explicitly set (with **`TextureTarget`**) then target is assumed to be in the vertex stream

Material Settings

```
PrtReceiver0 ... DiffuseCoef("tBase") WSNormals("tBump") Refraction(1.5)
                 ScatteringCoef(1.19, 1.62, 2.0) AbsorbtionCoef(0.021, 0.041, 0.071)
```

- **`DiffuseCoef()`** : Diffuse reflectance coefficient
  - Literal : (1.0, 1.0, 1.0)
  - Artist Editable Variable : Popup color picker in Maya
  - VertexColor : Use the vertex color
  - Texture Name : Use albedo map
- **`WSNormals()`** : Name of a World-Space normal map
  - If not specified, default to the geometric normals
- **`Refraction()`** : Index of refraction
- **`ScatteringCoef()`** : Reduced Scattering Coefficients
- **`AbsorptionCoef()`** : Absorption Coeffiecients

Only used for subsurface scattering

Configuring a PRT Blocker

```
PrtBlocker0
```

- No further settings necessary
- Blocker geometry is culled after PRT simulation and never makes it to the runtime
- Blockers can be used in a few different ways…

Two Pass Preprocessor

- First pass
  - Read raw geometry and parse shader headers
  - Run PRT Simulation
  - Save PRT results
- Second Pass
  - Read PRT results
  - Compress
  - Convert mesh data to runtime format
- This allows us to:
  - Reuse results
  - Run simulator on high-res geometry but apply results to low res-geometry

Multiple PRT “Materials”

```
PrtReceiver0
PrtReceiver1
```

- Multiple simulations enabled in a header, referenced by index
- We refer to these as PRT Materials
- Preprocessor groups geometry by PRT Material and PRT Type

```
for (int index = 0; index < MAX_PRT_MATERIALS; index++)
{
   Blockers = FindAllBlockerMeshes(index);
   Receivers = FindAllReceiverMeshes(index);
   PRTResults = LaunchPRTSimulator(Blockers, Receivers);
   SavePRTResults(PRTResults);
}
```

Demo Goals

- PRT/SH Lighting for everything
- Illumination from indoor and outdoor sources
- Simple animation to demonstrate subsurface scattering at different scales

Demo

How we did it…

- Exporting scene elements
- Indoor/Outdoor illumination
- Light grouping
- Animating the statue
- Conserving memory
  - Geometry instancing
  - Compression

Exporting the Scene as One Object

- Each mesh is assigned a PRT Receiver shader
- Every object shadows every other object
- Long simulation time
- If one mesh is wrong, the entire scene must be re-exported and the simulation re-run
- Not very modular (in terms of scene objects)
- Just not a good idea

Exporting Each Object Separately

- Each object is a receiver and is exported with a group of blockers
- Objects only shadowed by explicitly chosen blockers
- Shorter simulation time per-object
- Objects can be modified and re-simulated separately

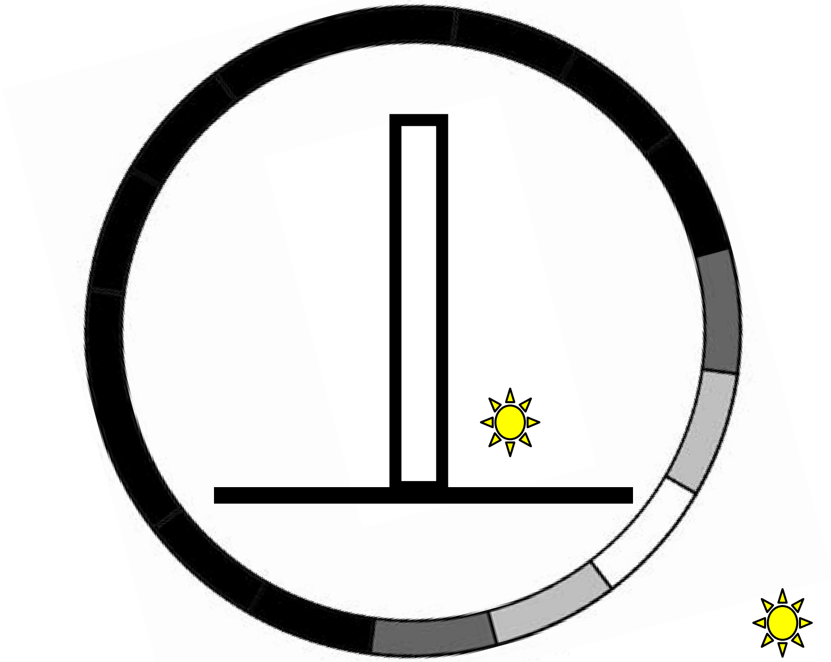
Blockers and Receivers

- Think of the scene as a bunch of discreet receivers:
  - Pedestal, Columns, Floor, Ceiling, etc
- Each receiver has a list of blockers that cast shadows onto it
- Each receiver is exported/preprocessed along with all of it’s blockers

Choose Blockers Carefully

- Decide which blockers shadow your receivers
- Incoming light is on a distant sphere (like an environment map)
  - Light sources may not get between a receiver and it’s blockers
- Visualize the bounding sphere around a receiver and it’s blockers

Beating the Distant Sphere

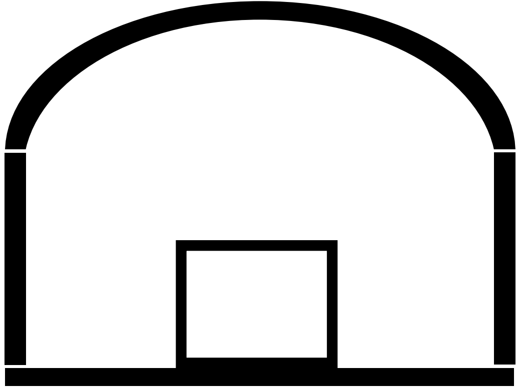


- Column exported with no blocker
- Floor exported with column blocker
- By explicitly choosing the blockers for each receiver you can reduce this limitation

This works most of the time

- The light source probably won’t go under the floor anyway, so the floor doesn’t need to block the column
- The shadow cast by the column onto the floor won’t be exactly correct
  - But it will look good enough (these are low frequency shadows anyway)
- This still isn’t good enough for indoor scenes…

Indoor & Outdoor Light Sources



- We want to have lights inside and outside the structure
- Explicitly choosing a single set of blockers isn't flexible enough
- Good outdoor blocker do not always make good indoor blockers
  - Either wrong for indoor lights or wrong for outdoor lights

Multiple Blocker Groups

- A receiver can have multiple groups of blockers
  - Blockers for outdoor lights
  - Blockers for indoor lights
- Multiple PRT materials in shader header
  - Receivers have 2 PRT Receiver Materials
  - Blockers have up to 2 PRT Blocker Materials
- Simulator gets launched twice once for each Receiver/Blocker Material Group

Indoor/Outdoor Blockers

- Indoor Blocker’s Shader
  - Two PRT Materials in header
  - Both Materials are of type “Blocker”

```
PrtBlocker0
PrtBlocker1
```

- Outdoor Blocker’s Shader
  - One PRT Material in header
  - Material is of type “Blocker”

```
PrtBlocker1
```

- Indoor/Outdoor Receiver Shader
  - Two PRT Materials in header
  - Both of type “Receiver”

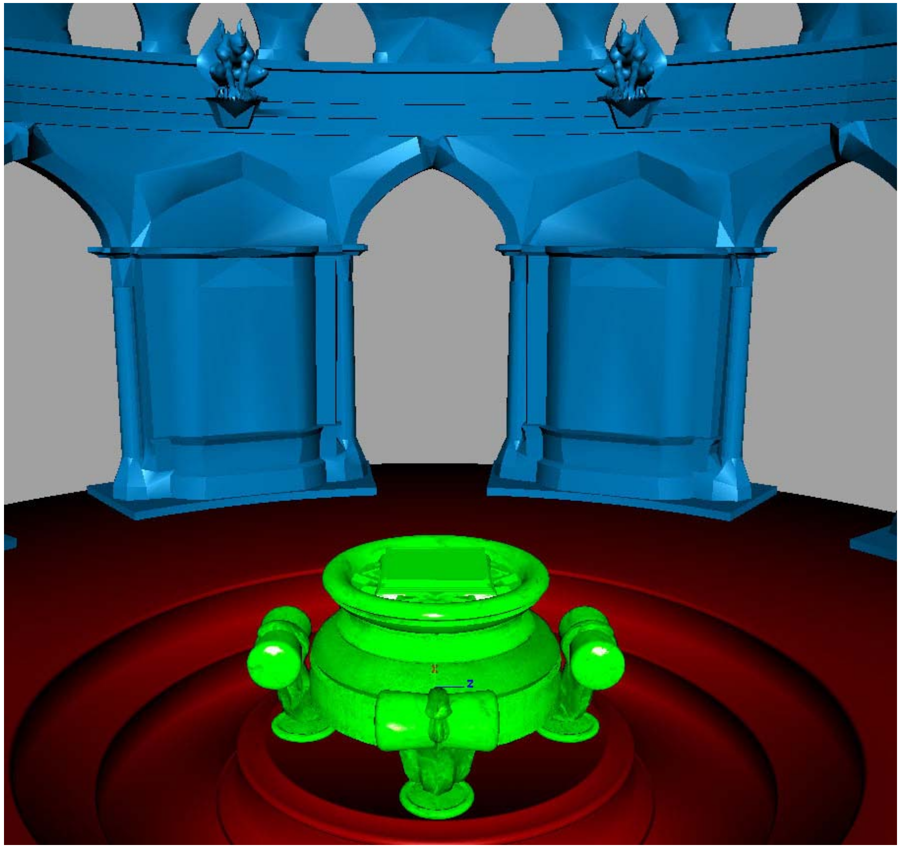
```
PrtReceiver0
PrtReceiver1
```

Indoor/Outdoor Simulation

- Preprocessor sends Floor to the simulator twice
- First simulation (PRT Material Index = 0)
  - Floor is receiver
  - Pedestal is blocker
- Second Simulation (PRT Material Index = 1)
  - Floor is receiver
  - Pedestal, Walls, Ceiling are blockers
- At run time the shader computes:
  - Integrate Outdoor illumination with “Outdoor” transfer coefficients
  - Integrate Indoor illumination with “Indoor” transfer coefficients
  - Add results for combined indoor and outdoor diffuse reflection

Indoor/Outdoor Shaders

- Sounds more complicated than it is…
  - Floor: Receiver Shader (2 PRT Materials in header)
  - Outdoor blockers: Blocker Shader (1 PRT Material in header)
  - Indoor blockers: Blocker Shader (2 PRT Materials in header)



Results of Outdoor Illumination Only

Results of Indoor Illumination Only

Outdoor + Indoor Illumination

Light Grouping

- Artist groups lights for each PRT Material
  - One group of lights used to generate outdoor lighting environment
  - One group of lights used to generate indoor lighting environment
- It is possible to have lights that change from outdoor lights to indoor lights
  - As light crosses some positional threshold, blend it out of one group and into another
  - We don’t currently do this…

Quick Note on Many Lights

- You can use many lights if you like
  - They are all just summed on the CPU to a single spherical signal anyway
- But you really don’t want to use too many lights
  - They’re low frequency so too many lights on a single object makes the object look full bright… no shadows, etc.

Animation

- **`BindFrame()`**
  - Model is pre-transformed to a specific frame of animation before it's sent to the simulator
- Multiple **`BindFrames`**
  - Multiple poses sent to simulator
    - Multiple PRT Materials defined in shader
  - Not just for object animation, can capture ***Material*** animation too
    - Blended at runtime in the shader

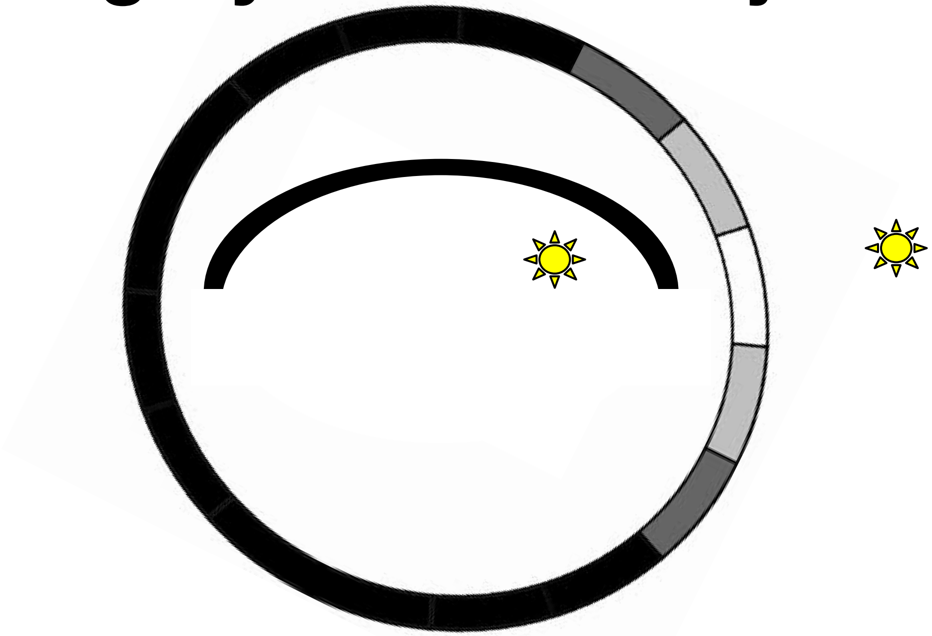
Animating the Statue

- Two PRT Materials enabled in Statue’s shader
- Each PRT Material defines different Subsurface Scattering Coefficients
  - First Material configures simulator to compute transfer for full scale model
  - Second Material configures simulator to compute transfer for small scale model
  - Results LERP’d in shader: weighted linear average based on current frame of animation

Instancing

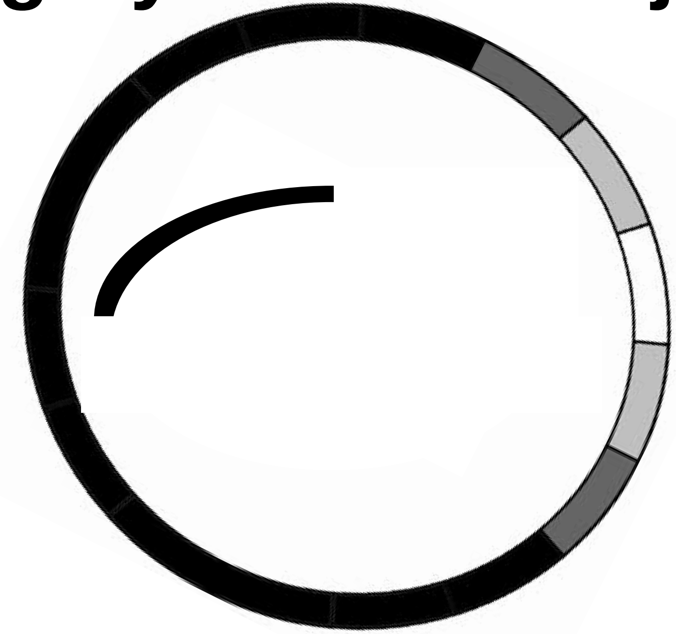
- Instancing autonomous objects is straight forward
  - Only store PRT results of one instance
  - Apply object’s inverse world transform to lighting environment to keep lighting environment and PRT in the same space
- Instancing symmetric pieces of an object can be useful…
  - Makes lighting concave objects a little easier

Instancing Symmetric Object Pieces



- This is fine for outdoor lights but indoor lights won’t work
- Using multiple blocker groups won’t solve this problem because the object is blocking itself
- Instead this object could be chopped along it’s line of symmetry and drawn twice at runtime

Instancing Symmetric Object Pieces



- Now indoor lights can properly illuminate this instanced piece
- Export the other half as an outdoor blocker to keep outdoor lights working too

Compression

- Use CPCA!
  - Saves massive amounts of video memory for both per-vertex and per-texel PRT
  - It’s fully supported by the D3DX PRT API
- We have found that compressing PRT textures using DXT5 works… sometimes
  - Instead of compressing every PRT texture on an object try compressing a few
  - Experiment, try compressing every other PRT texture
  - This worked for us, your mileage may vary

Conclusion

- Preprocessor
  - Flexible, Shader Based
  - Receivers & Blockers
  - Allows multiple PRT simulations per-shader
- Demo
  - PRT used for all lighting
  - Worked around some PRT limitations
    - Indoor/Outdoor lights
- Take the ideas you liked and add them to your own PRT tools

Thank you!

- Peter-Pike Sloan
- Robin Green
- Dan Roeger

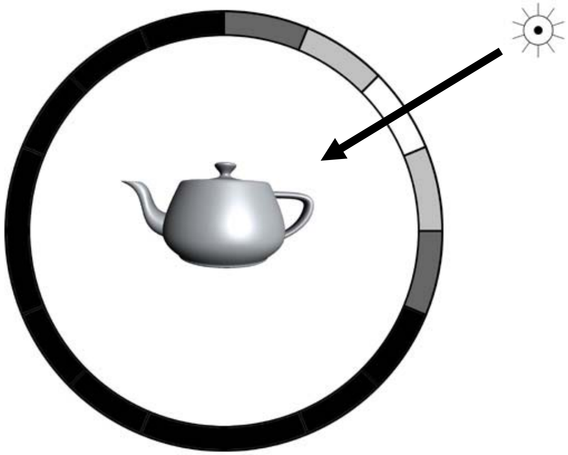
Extensions

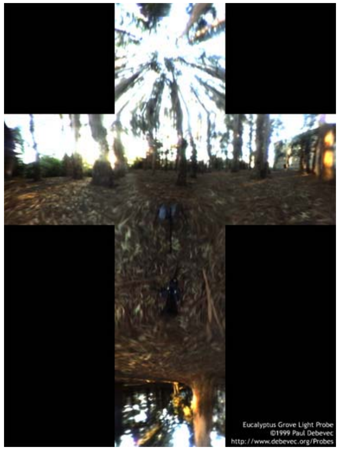
- Other (Non-SH) transfer functions
- SH emitters
- General emitters
- Special emitters
- Mixing General/Special/SH emitters all in the same transfer vector

Normal Maps

- If they’re bigger than PRT maps…you’ll down-sample them right? But what happens to gutter regions of normal map? Badness.
- Start with 1:1 mapping of normal map texels to PRT texels.
- Differentiate between runtime maps and preprocess maps.

Spherical Light Signal

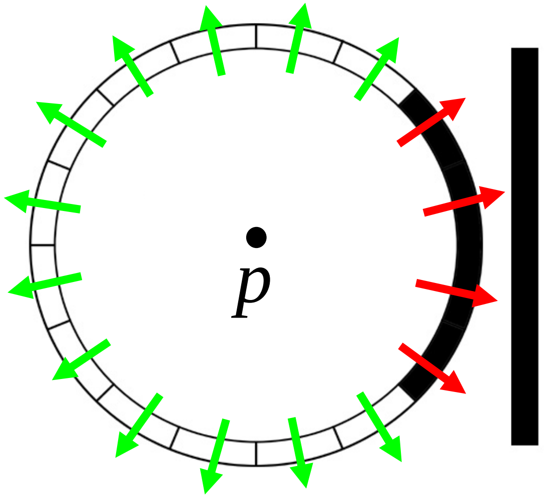




Eucalyptus Grove Light Probe
©1999 Paul Debevec
http://www.debevec.org/Probes

- This is traditionally implemented using a cubemap
- Complex lighting environments can be captured (not just discrete point lights)
  - Captured at a single point but used at many points
  - Lighting environment is infinitely far away

Spherical Visibility Signal



- Visibility is stored as a spherical signal
- This does not encode blocker's distance from 'p'
- Light source can not get between 'p' and the blocker

Storing Spherical Signals

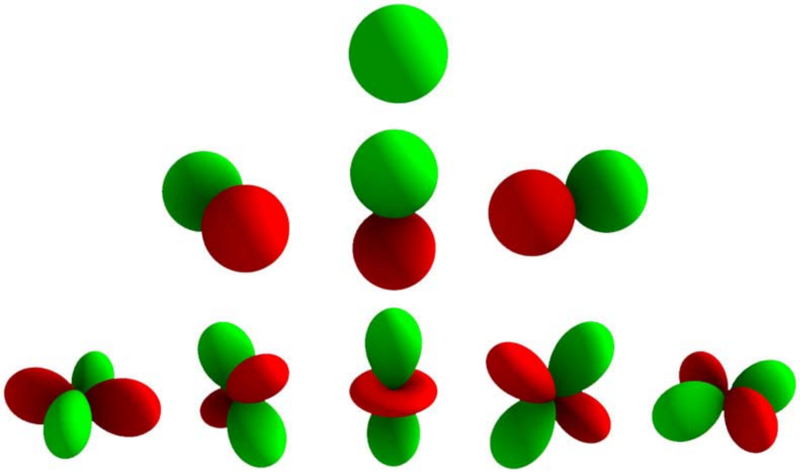
Incoming light : stored once per-lighting environment

Visibility : stored at every point ‘p’ on surface

Hemisphere Cosine : stored at every point ‘p’ on surface

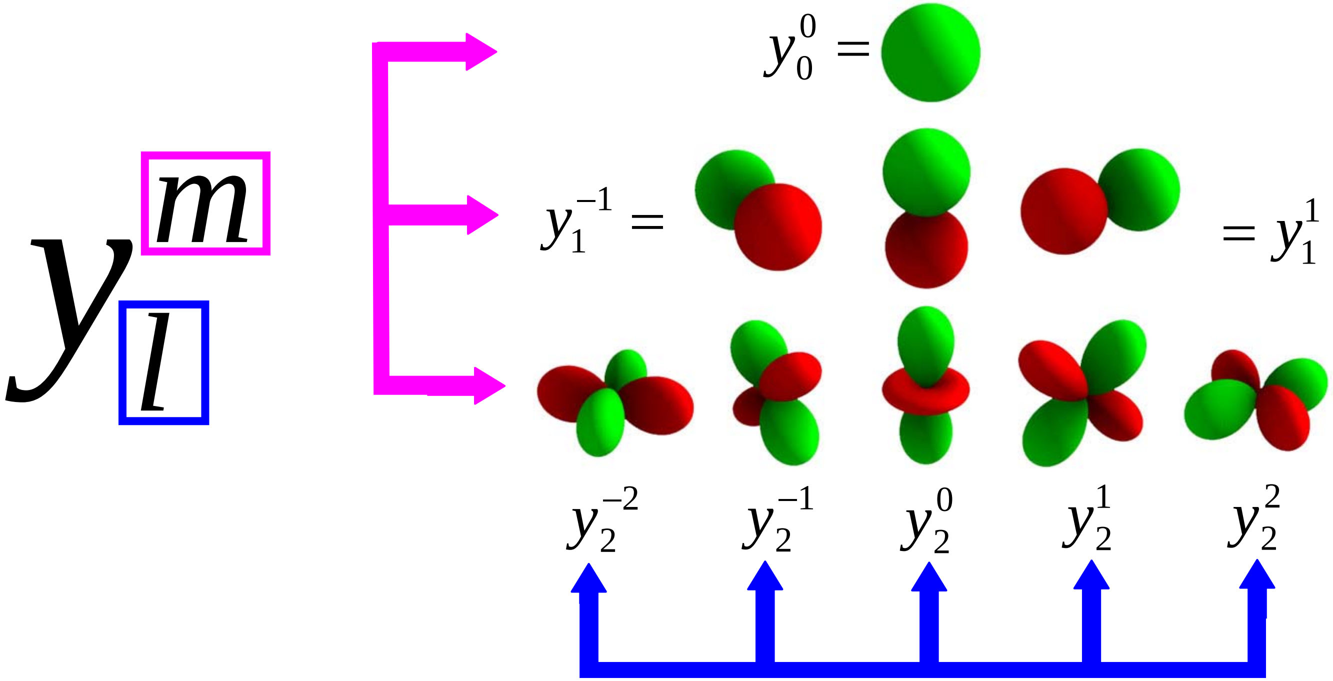
- Storing a cubemap for every point p on the model is not feasible
- Signal can be efficiently approximated with Spherical Harmonic basis functions

Spherical Harmonics



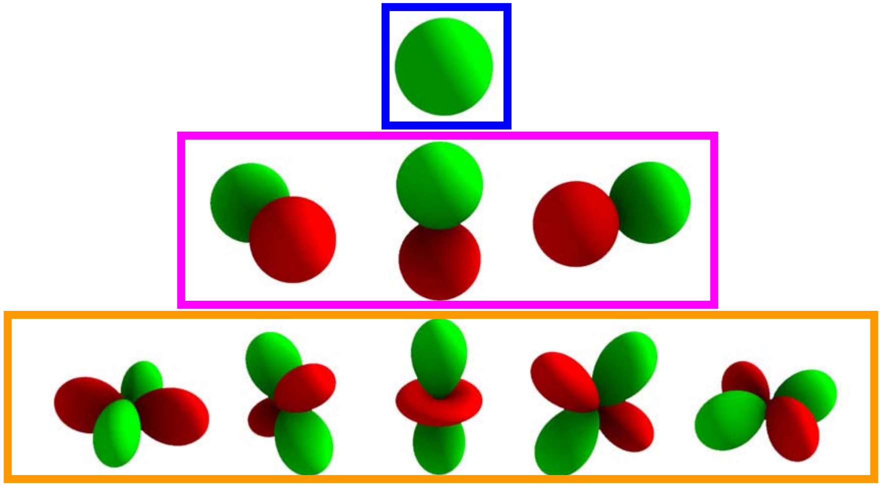
- Infinite Series of Spherical Functions (the first 9 functions are shown here)
- If we use a bunch of these as basis functions, we can compactly approximate a low frequency spherical signal

SH Notation



- m designates the band
- l is the index within the band

SH Basis Functions



$$< \boxed{C_0,}\ \boxed{C_1,\ C_2,\ C_3,}\ \boxed{C_4,\ C_5,\ C_6,\ C_7,\ C_8,\ C_9,}\ \ldots\ C_n >$$

- Series is infinite
  - Choose a range that fits storage and approximation needs
  - Each function in the truncated series is assigned to an element in a vector.
- Each element stores its associated SH function’s contribution to the overall signal (basis weight)
  - It’s like building your original spherical signal out of a fixed set of scaled, predefined spherical signals
  - The larger the “fixed set” the closer the approximation will be

Math with SH Basis Functions

- Adding two SH Functions
  - Add two vectors
- Integrate two SH Functions
  - Dot product of two vectors

Spherical Harmonic Lighting

- Object’s incident light is stored using SH basis functions (L)
- L maybe sampled directly using a Normal Vector or…
- If you’ve pre-computed transfer functions, you can solve the rendering equation directly!

Pre-Computed Radiance Transfer

- Object's radiance transfer is stored using SH basis functions (T_i)
- Lighting environment is computed once for the entire object
- Lighting a point on the object is: $L \bullet T_p$