Monte Carlo Integration: Direct Lighting

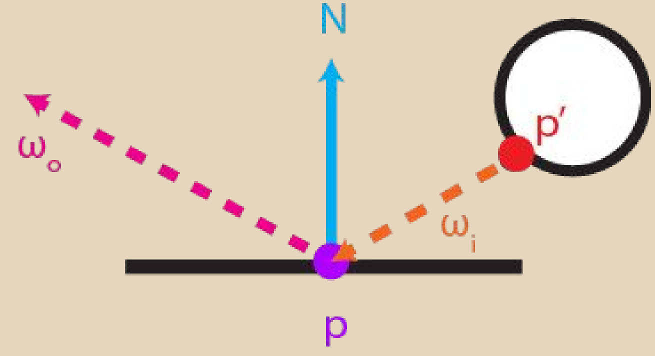
Adam Mally
CIS 560 Spring 2015
University of Pennsylvania

Monte Carlo Path Tracer vs Ray Tracer

- MCPT attempts to solve the lighting equation for all visible points in the scene
- Cast many, many rays per pixel in order to gather enough lighting information
- Effects like caustics, soft shadows, anti-aliasing, and depth of field are essentially free
- Without parallelization and other optimizations, converges to a useful image incredibly slowly

- RT casts one ray per pixel (excluding anti-aliasing)
- Makes physically incorrect assumptions about light transportation in order to ensure a fast run time
- Must add special computations to support/fake effects like caustics and depth of field
- Produces a usable image in a relatively short time, even if the image is not physically accurate

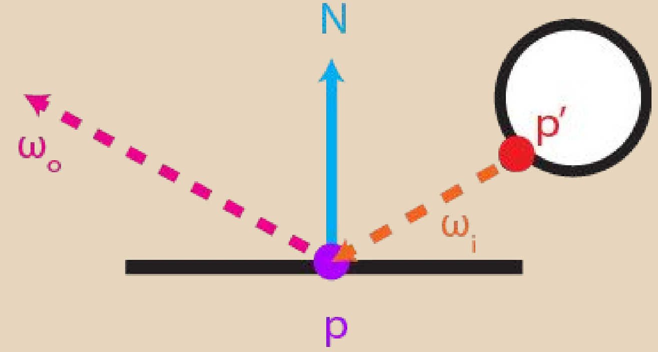
Light Transport Equation - Integral



$L_o(p,\omega_o) = L_E(p,\omega_o) + \int_S f(p,\omega_o,\omega_i)L_i(p,\omega_i)\ V(p',p)\ \mathrm{absdot}(\omega_i,\ N)\ d\omega_i$

- L_o: radiance from a point p and direction ω_o.
- L_E: emitted radiance from a point (nonzero only if p is a light source; all other light is *reflected*)
- f: BRDF of surface at p
- L: radiance at some point x in direction ω_i.
- V: visibility between p’ and p (1 when unobstructed, 0 otherwise)
- We incorporate the dot product of ω_i and N due to Lambert’s law
- S is the sphere of all possible ω_i that can reach p

Light Transport Equation - Sum



$L_o(p,\omega_o) = L_E(p,\omega_o) + \Sigma_\omega\ (f(p,\omega_o,\omega_i)L_i(p,\omega_i)\ V(p',p)\ \mathrm{absdot}(\omega_i,\ N)\ d\omega_i)$

- L_o: radiance from a point p and direction ω_o.
- L_E: emitted radiance from a point (nonzero only if p is a light source; all other light is *reflected*)
- f: BRDF of surface at p
- L: radiance at some point x in direction ω_i.
- V: visibility between p’ and p (1 when unobstructed, 0 otherwise)
- We incorporate the dot product of ω_i and N due to Lambert’s law
- For some discrete subset of S, choose n ω to sample and average their reflected light energies
  - The more samples taken, the closer the average is to being physically accurate

Expected Value

- We use Monte Carlo integration to estimate the expected value of the Light Transport Equation (which is an integral)
- The expected value of of a function f is the average value of f over some distribution of values $p(x)$ over its domain D
  - $E_p[f(x)] = \int_D f(x)p(x)dx$
  - Note that $p(x)$ must integrate to 1 over the domain D since it is a probability distribution function
  - This means $p(x)$ weights the result of $f(x)$ for any given x while evaluating this integral

Monte Carlo Estimator

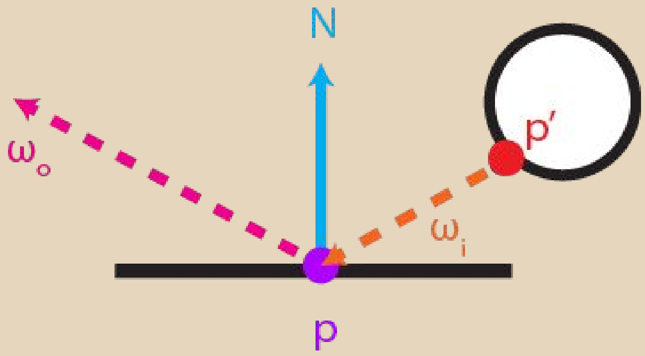
- Let's say we want to evaluate some integral $\int_a^b f(x)dx$
- Given a set of **uniform** random variables X_i in the range $[a,b]$, the Monte Carlo estimator says the expected value of our estimator is **equal to the integral of *f(x)***
- Our estimator is:
  - $F_N = (b - a)/N * \Sigma^N_{i=1} f(X_i)$
- Whis is this the case?
  - $p(x)$ must equal $1/(b-a)$ since it must be a constant and integrate to 1 over the domain $[a,b]$
  - With algebraic manipulation, we see the following:
  - $E[F_N] = E[(b - a)/N * \Sigma^N_{i=1} f(X_i)]$
  - $E[F_N] = (b - a)/N * \Sigma^N_{i=1} E[f(X_i)]$
  - $E[F_N] = (b - a)/N * \Sigma^N_{i=1} \int_a^b f(x)p(x)dx$
  - $E[F_N] = 1/N * \Sigma^N_{i=1} \int_a^b f(x)dx$
  - $E[F_N] = \int_a^b f(x)dx$

Monte Carlo Estimator

- What if the set of random variables we use to estimate *E[f(x)]* is **not uniform?**
  - e.g. if we're using **importance sampling**
- Our estimator now becomes:
  - $F_N = 1/N \Sigma^N_{i=1} f(X_i)/p(X_i)$
  - Our only limitation is that $p(X_i)$ be nonzero wherever $f(X_i) > 0$
- Let's prove that this still evaluates to the integral of *f(x)*:
  - $E[F_N] = E[1/N * \Sigma^N_{i=1} f(X_i)/p(X_i)]$
  - $E[F_N] = 1/N * \Sigma^N_{i=1} \int_a^b (f(x)/p(x))p(x)dx$
  - $E[F_N] = 1/N * \Sigma^N_{i=1} \int_a^b f(x)dx$
  - $E[F_N] = \int_a^b f(x)dx$

What is a BRDF?

- A function that evaluates the energy emitted along ray ω_o given a point of intersection **p** and the direction from which the incoming light emits, ω_i
- Entirely dependent on the attributes of the material sampled at point p
- Examples:
  - A Lambertian BRDF simply takes the material's base color and divides it by π
    - Since we integrate over the entire *hemisphere* of visible directions from **p**, we need to divide by π so that the integral evaluates to the material's base color
  - A BRDF describing perfect specular reflection would be a discontinuous function that returns 0 (black) for all pairs of ω_o and ω_i that are not reflections of one another around the normal at **p**



Monte Carlo Path Tracer - Naive Method

- For each pixel, cast N rays through N different points within the pixel
- For each camera ray, get the intersection with the scene
- If the surface is a light source, return its emitted light.
- If the surface is **not** a light source, randomly reflect the camera ray within the hemisphere of possible directions around the point of intersection
  - Recursively trace this ray until you reach some depth limit
  - The recursive tracing is necessary to solve for the $L_i(p, \omega_i)$ portion of the LTE
- This method is the least biased solution to the Light Transport Equation, but it will also take the longest to converge to a usable image
  - i.e. you'll need to take so many ray samples per pixel that the cost is too prohibitive to use

Monte Carlo Path Tracer - Importance Sampling

- There are several methods we can employ to reduce the number of samples needed to produce a "converged" image
- **Direct Light Sampling**: Light sources are the most important elements in a rendered scene; with no light, there's no image!
  - For some subset of the rays ω_i , select directions such that each ω_i intersects a given light source at some point
- **BRDF Sampling**: Sample ray directions that have a higher contribution to the color reflected along ω_o
  - Very useful if you have a BRDF with a very narrow set of contributing rays, e.g. a perfectly specular reflective surface

Estimating Direct Lighting

- One method of importance sampling is to estimate how much light travels from an emissive surface and **directly** reaches a given point in the scene and is reflected back along ω_o
- For the purposes of illustration, let's assume there is exactly one light in this scene

$$L_D(p,\omega_o) = \int_S f(p,\omega_o,\omega_i)L_d(p,\omega_i)\mathrm{absdot}(\omega_i, N)\, d\omega_i$$

- To write this as an estimate, we need to compute:

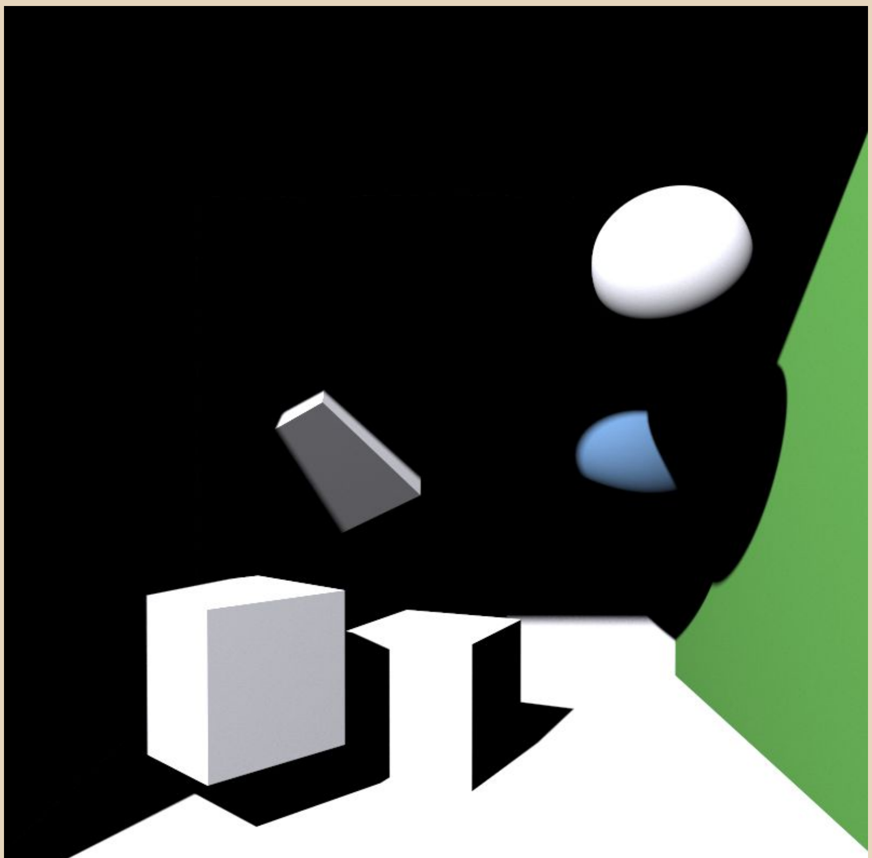
$$L_D(p,\omega_o) = 1/N\ \Sigma^N_j\ (f(p,\omega_o,\omega_j)L_d(p,\omega_j)\mathrm{absdot}(\omega_j, N)/p(\omega_j))$$

  - $p(\omega_j)$ is the probability that ω_j will contribute light
  - $f(p,\omega_o,\omega_j)$ is the BRDF of our point
  - $L_d(p,\omega_j)$ is the light directly reaching p from direction ω_j

- We want to reduce the variance in our estimates, so we will use importance sampling to choose our various ω_j

For clarity: Direct lighting example

Direct lighting ONLY



Direct + indirect lighting

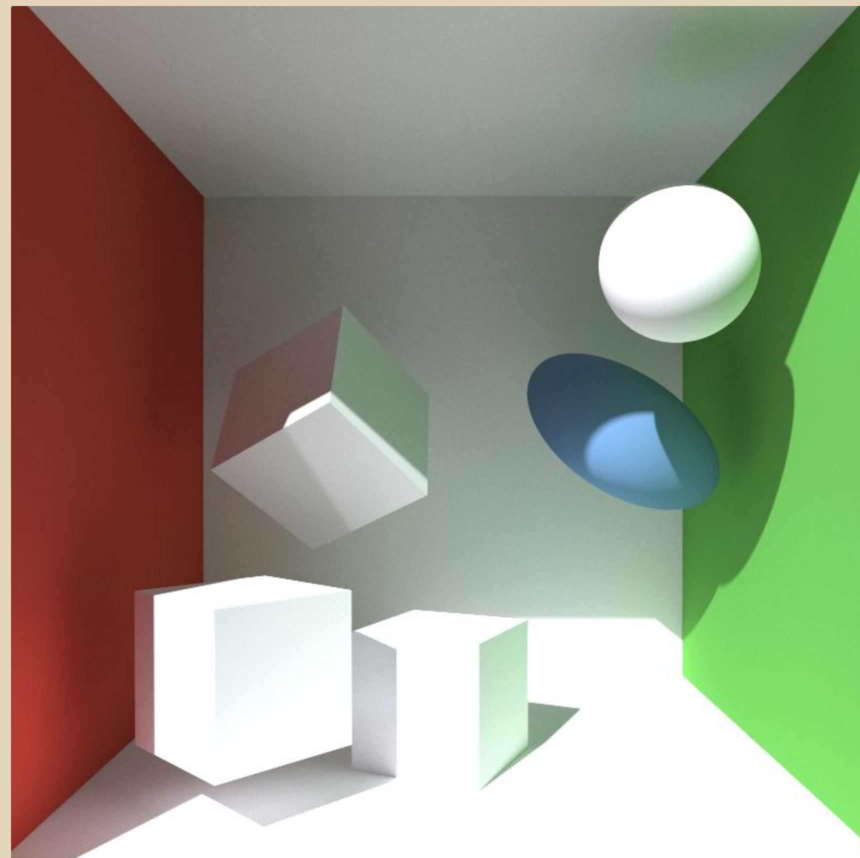


Image source: yiningkarlli.com

Multiple Importance Sampling for DLE

- Since it would be difficult to find a function that matches the PDFs of the light source and the BRDF of our surface point, we will importance sample each PDF separately and weight our results
- Depending on the characteristics of the light and BRDF, one of these sampling methods may be far more effective than the other
- Examples:
  - When a BRDF is more specular, sampling only the **light's PDF** makes it less likely that a large light will contribute to its color (left image)
  - When a BRDF is more diffuse, sampling only the **BRDF's PDF** makes it less likely that small lights will contribute to its color (right image)

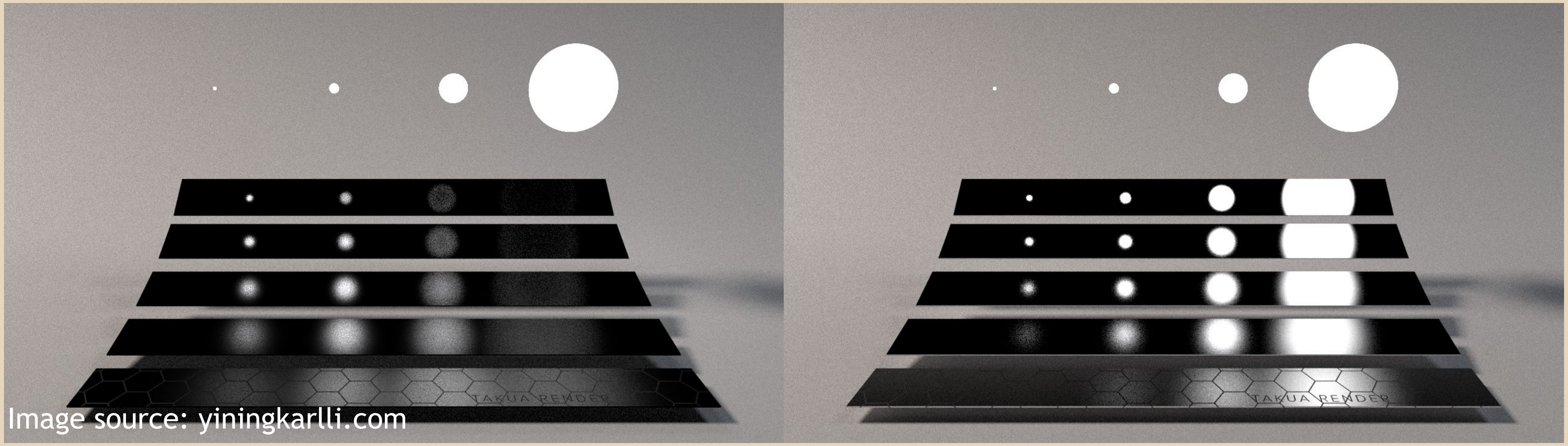


Image source: yiningkarlli.com

Multiple Importance Sampling for DLE

- Since each sampling method is effective in different scenarios, they will produce a high amount of **variance** in our image if not weighted properly
- Since we are sampling two functions (let's call them $f(x)$ and $g(x)$) and have a different importance sampling strategy for each, normally we'd have to choose one strategy over the other
  - Choosing a single strategy will often give poor results for $f(x)$ or $g(x)$
- Simply taking results from our estimator using both sampling methods and averaging the results won't work; variance is *additive*, so it can't be "averaged out"
- So, we need to weight our samples so that they don't contribute too much to our result when the sampling density does not match the shape of our integrand
- Two such weighting methods are known as the **balance heuristic** and the **power heuristic**

MIS: The Balance Heuristic

- If two sampling functions p_f and p_g are used to estimate the value of $\int f(x)g(x)dx$, our Monte Carlo estimator becomes:
  - $1/n_f \Sigma^{n_f}_{i=1} f(X_i) g(X_i) w_f(X_i) \;/\; p_f(X_i) + 1/n_g \Sigma^{n_g}_{j=1} f(Y_j) g(Y_j) w_g(Y_j) \;/\; p_g(Y_j)$
    - n_f is the number of samples taken from the distribution method p_f, and n_g is the number of samples taken from the distribution method n_g
    - w_f and w_g are special weighting functions chosen such that the value of this estimator is the value of the integral $\int f(x)g(x)dx$
- If we use the **balance heuristic** as w_f and w_g, we get:
  - $w_f = n_f p_f(x) / (n_f p_f(x) + n_g p_g(x))$
  - $w_g = n_g p_g(x) / (n_f p_f(x) + n_g p_g(x))$
- Why does this work? Let's see on the next slide

MIS: The Balance Heuristic

- $w_f = n_f p_f(x) / (n_f p_f(x) + n_g p_g(x))$
- $w_g = n_g p_g(x) / (n_f p_f(x) + n_g p_g(x))$
- Let's consider a case in which the values of $p_f(x)$ and $f(x)$ are low, but $g(x)$ is high
  - With standard importance sampling, $f(x)g(x)/p_f(x)$ will have a high value because $p_f(x)$ is low and $g(x)$ is high
  - With the balance heuristic using $f(x)$, our importance sampling formula becomes:
  - $f(x)\mathbf{g}(x)n_f p_f(x) / (p_f(x) * (n_f p_f(x) + n_g p_g(x)))$
    $= f(x)\mathbf{g}(x)n_f / ((n_f p_f(x) + n_g p_g(x)))$
- As long as $p_g(x)$ has a distribution that roughly matches $g(x)$, then the denominator won't ever be too small when $g(x)$ is large and $p_f(x)$ is small
- The **power heuristic** is a modification of the balance heuristic that reduces variance even further
- Just square the numerator and both halves of the denominator of the balance heuristic and you've got the power heuristic
- For a proof of why this is more effective, read Eric Veach's thesis

Estimating Direct Lighting: Light PDF Sampling

- Now, back to actually estimating the direct lighting at a point

$L_D(p,\omega_o) = 1/N\ \Sigma^N_j\ (w(\omega_j)f(p,\omega_o,\omega_j)L_d(p,\omega_j)\mathrm{absdot}(\omega_j,\ N)/p(\omega_j))$

1. Take one sample using the light source's sampling distribution
   a. Ask the light source for a random point on its surface **p'**
   b. Create a ray from **p** to **p'**. This will be our ω_j
   c. Ask the light for its PDF (we'll cover this shortly)
   d. Check that the path from **p** to **p'** is unobstructed (if it is, our received light will be 0)
   e. Calculate the light reaching **p** along ω_j (this is $L_d(p,\omega_j)$)
2. If the PDF is nonzero and the light emitted along ω_j is nonzero, then we know we can evaluate this function
3. Now evaluate $f(p,\omega_o,\omega_j)$
   a. If this BRDF is nonzero, then we can ask **p**'s material for its PDF as well so we can use it in our power heuristic
   b. We can compute the power heuristic weight now that we have the light_pdf and brdf_pdf
4. Our (first) estimated light is equal to the formula at the top of the page for the ω_j computed using a sample on the light source

Estimating Direct Lighting: BRDF PDF Sampling

- Part two of estimating the direct lighting at a point

$L_D(p,\omega_o) = 1/N\ \Sigma^N_j\ (w(\omega_j)f(p,\omega_o,\omega_j)L_d(p,\omega_j)\mathrm{absdot}(\omega_j,\ N)/p(\omega_j))$

1. Take one sample using the BRDF's sampling distribution
    a. Ask p's BRDF to create a ω_j from its PDF given ω_o
    b. Store ω_j and the PDF of ω_j
    c. Sample the BRDF using this ω_j and ω_o
2. If the PDF of the BRDF is nonzero and the light emitted along ω_j is nonzero, then we know we can evaluate this entire function
3. Now evaluate $L_d(p,\omega_j)$
    a. If the PDF of our light is nonzero given ω_j , then we want to compute the power heuristic weight using the brdf_pdf and the light_pdf
4. If wj intersects the chosen light source at all, compute the portion of light from the source that travels along ω_j; this is $L_d(p,\omega_j)$
5. Our (second) estimated light is equal to the formula at the top of the page for the ω_j computed using a sample on the BRDF
6. Sum the result of the previous slide with this slide's result for the final direct lighting computation

The PDF of a shape relative to a point

- We need to compute the portion of the hemisphere at p that can see the light source (or, more generally, any shape we want to sample directly)
- We begin by computing the PDF relative to the surface area of the shape
  - PDF_{area} = 1/area
- We need to convert this to a PDF relative to the shape's subtended solid angle with respect to **p**
  - $d\omega_j/dA = cos(\theta)/r^2$
    - θ is the angle between ω_j and the light's surface normal at the point where ω_j intersects it
    - r^2 is the distance between the point on the shape and **p**
- We need to divide 1/area by this factor to compute our solid angle PDF:
  - pdf = $r^2/(cos(\theta)*area)$
- Let's use an intuitive example for why this works:
  - $d\omega_j = dAcos(\theta)/r^2$
  - If the surface's normal is parallel to ω_j, then θ is 0. If r = 1, then $d\omega_j$= dA, and the equation holds
  - As the surface rotates away from ω_j, its solid angle decreases with cos(θ)

BRDFs: World vs Local Space

- In general, most contemporary path tracers evaluate the output of a BRDF using rays in “normal local space”
  - This treats the surface normal at the input intersection as **always** being <0 0 1>, so the input rays must be transformed accordingly
- Just like when computing the transformation matrix needed to use a normal map, one must calculate the **tangent** and **bitangent** at the point of intersection to make a world -> normal space matrix
- Let’s review the normal map slides...

Normal maps

- To compute the matrix that transforms from texture space to object space, we need three vectors: a normal, a tangent, and a bitangent
  - These form a local orthonormal space
  - The same concept as creating the orientation matrix for a camera
- We already have our normal, which is the one we'd use if there were no normal map
- The tangent corresponds to our local X axis, so it should align with the U axis of our texture
  - Spherical example: For all points except the very poles of our sphere, we can get the tangent by crossing <0,1,0> with our normal (remember to normalize the result)
- Likewise, the bitangent corresponds to our local Y axis, so it should align with the V axis of our texture
  - Sphere: Get bitangent by crossing our normal with our tangent
- Our transformation matrix can now be created:

$$\begin{vmatrix} T.x & B.x & N.x & 0 \\ T.y & B.y & N.y & 0 \\ T.z & B.z & N.z & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

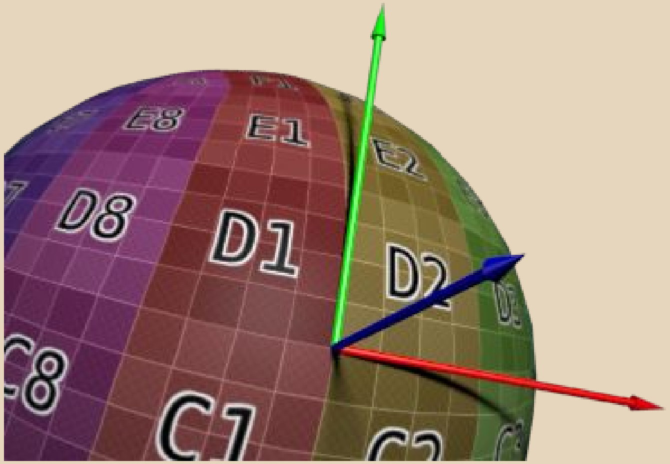


Image source: http://www.opengl-tutorial.org/wp-content/uploads/2011/05/NTBFromUVs.png

Normal maps

- Note that this matrix converts from world to local space, so we need to compute its **inverse**!
- Fortunately, because this is an orthogonal matrix (each column is perpendicular to one another) its inverse is equal to its transpose!
- So, the matrix we want is in fact:

$$\begin{vmatrix} T.x & B.x & N.x & 0 \\ T.y & B.y & N.y & 0 \\ T.z & B.z & N.z & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}^{-1} = \begin{vmatrix} T.x & T.y & T.z & 0 \\ B.x & B.y & B.y & 0 \\ N.z & N.z & N.z & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Tangents and bitangents based on UVs

- In a general case, such as a triangle, we want to be able to compute our tangent and bitangent regardless of UV mapping technique
- If we know three points **of a plane** on our object and the UVs at those points (which gives us two changes in positions and UVs), we can solve a system of linear equations to compute our tangent and bitangent:
  - $\Delta\text{Pos1} = \Delta\text{UV1.x} * T + \Delta\text{UV1.y} * B$
  - $\Delta\text{Pos2} = \Delta\text{UV2.x} * T + \Delta\text{UV2.y} * B$

- $B = (\Delta\text{Pos2} - \Delta\text{UV2.x} * T)/\Delta\text{UV2.y}$ OR $(\Delta\text{Pos1} - \Delta\text{UV1.x} * T)/\Delta\text{UV1.y}$
- $T = (\Delta\text{UV2.y}\Delta\text{Pos1} - \Delta\text{UV1.y}\Delta\text{Pos2}) / (\Delta\text{UV2.y}\Delta\text{UV1.x} - \Delta\text{UV1.y}\Delta\text{UV2.x})$