COMP3421

Global Lighting Part 2: Radiosity

Recap: Global Lighting

The lighting equation we looked at earlier only handled direct lighting from sources:

$$I = I_a \rho_a + \sum_{l \in \text{lights}} I_l \left(\rho_d (\hat{\mathbf{s}}_\mathbf{l} \cdot \hat{\mathbf{m}}) + \rho_{sp} (\hat{\mathbf{r}}_\mathbf{l} \cdot \hat{\mathbf{v}})^f \right)$$

We added an ambient fudge term to account for all other light in the scene.

Without this term, surfaces not facing a light source are black.

Global lighting

In reality, the light falling on a surface comes from everywhere. Light from one surface is reflected onto another surface and then another, and another, and...

Methods that take this kind of multi-bounce lighting into account are called global lighting methods.

Raytracing and Radiosity

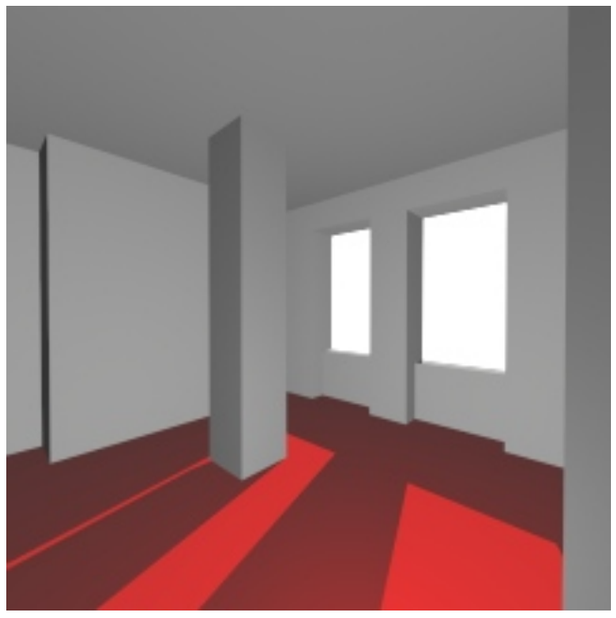
There are two main methods for global lighting:

- Raytracing models specular reflection and refraction.
- Radiosity models diffuse reflection.

Both methods are computationally expensive and are rarely suitable for real-time rendering.

Radiosity

Radiosity is a global illumination technique which performs indirect diffuse lighting.



direct lighting +
ambient

global
illumination

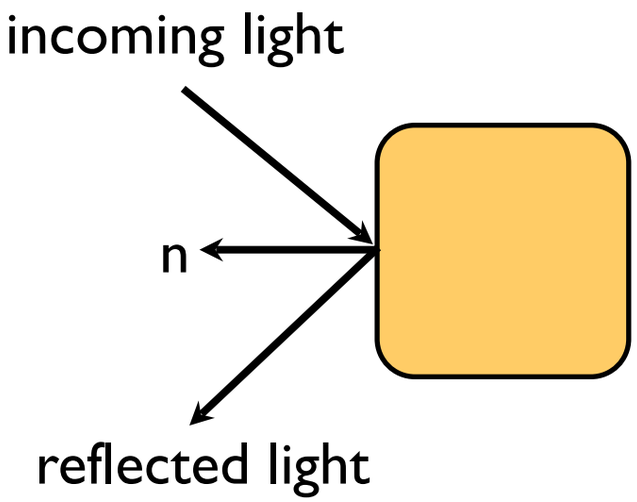
Radiosity

Direct lighting techniques only take into account light coming directly from a source.

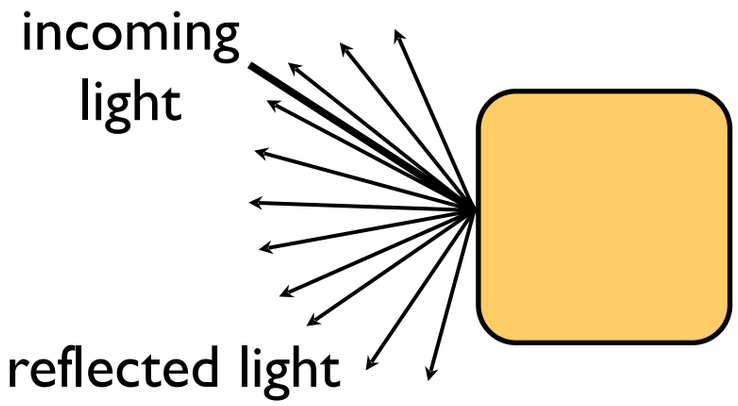
Raytracing takes into account specular reflections of other objects.

Radiosity takes into account diffuse reflections of everything else in the scene.

Ray tracing vs Radiosity

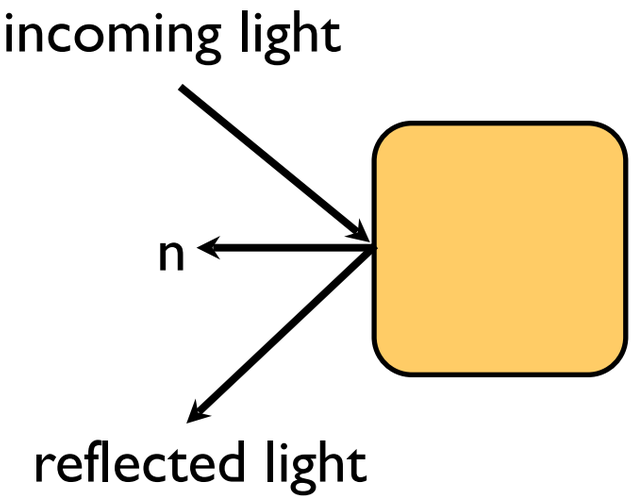


Specular reflection

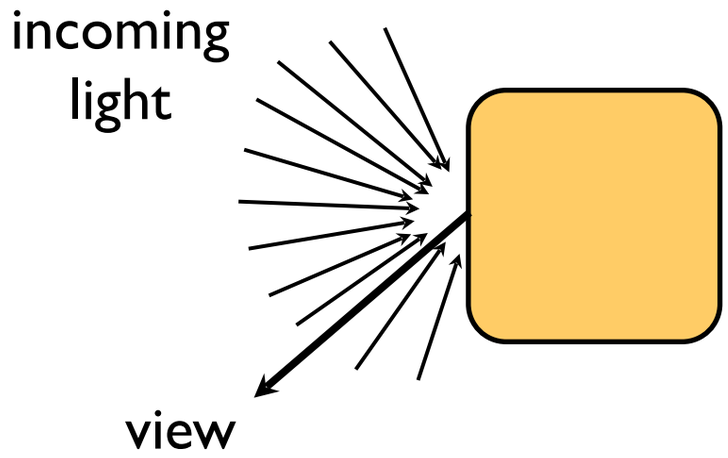


Diffuse reflection

Ray tracing vs Radiosity



Specular reflection



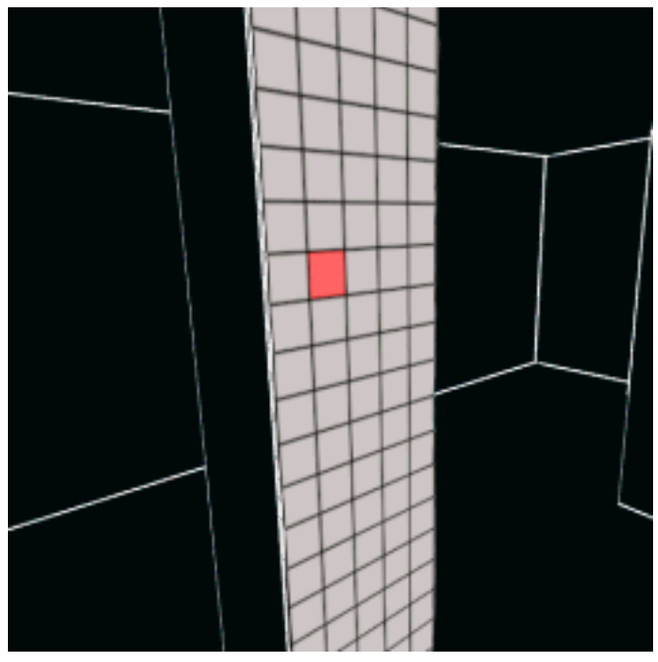
Diffuse reflection

Finite elements

We can solve the radiosity problem using a finite element method.

We divide the scene up into small patches.

We then calculate the energy transfer from each patch to every other patch.



Energy transfer

The basic equation for energy transfer is:

$$\text{Light output} = \text{Light emitted} + \rho * \text{Light input}$$

where ρ is the diffuse reflection coefficient.

Energy transfer

The light input to a patch is a weighted sum of the light output by every other patch.

$$B_i = E_i + \rho_i \sum_j B_j F_{ij}$$

B_i is the radiosity of patch i

E_i is the energy emitted by patch i

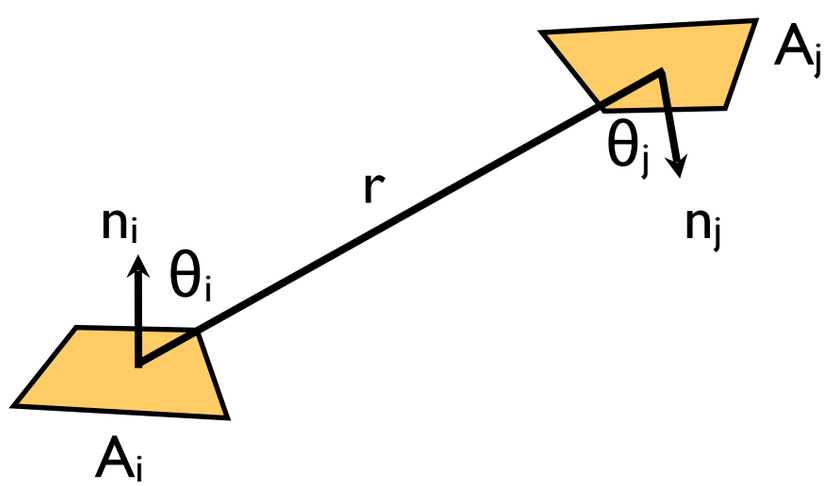
ρ_i is the reflectivity of patch i

F_{ij} is a form factor which encodes what fraction of light from patch j reaches patch i.

Form factors

The form factors Fij depend on

- the shapes of patches i and j
- the distance between the patches
- the relative orientation of the patches

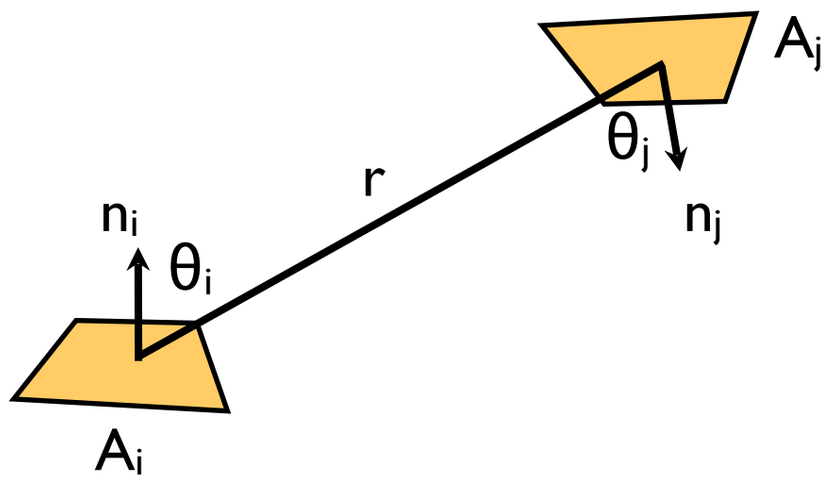


Form factors

Mathematically:

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi r^2} dA_j dA_i$$

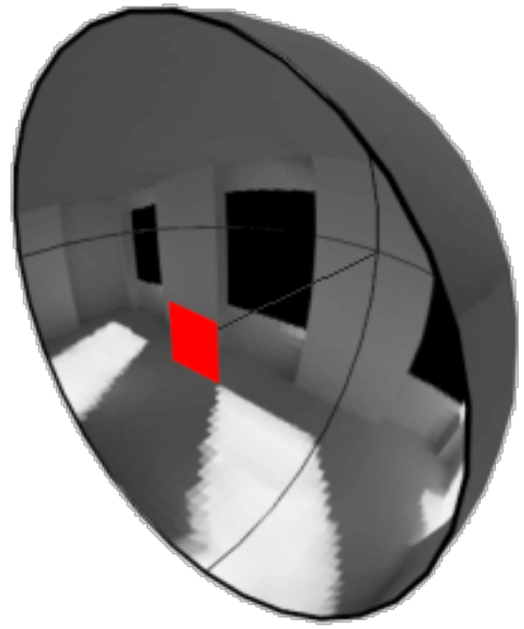
Calculating form factors in this way is difficult and does not take into account occlusion.



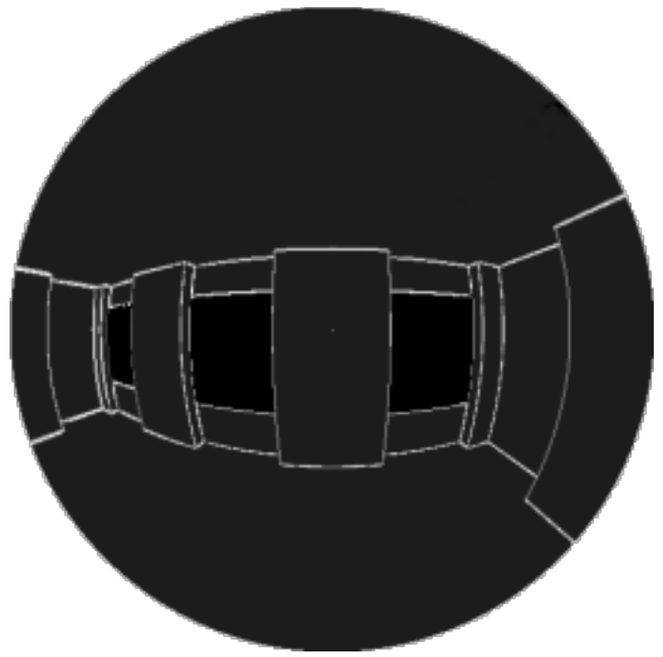
Nusselt Analog

An easier equivalent approach:

1. render the scene onto a unit hemisphere from the patch's point of view.

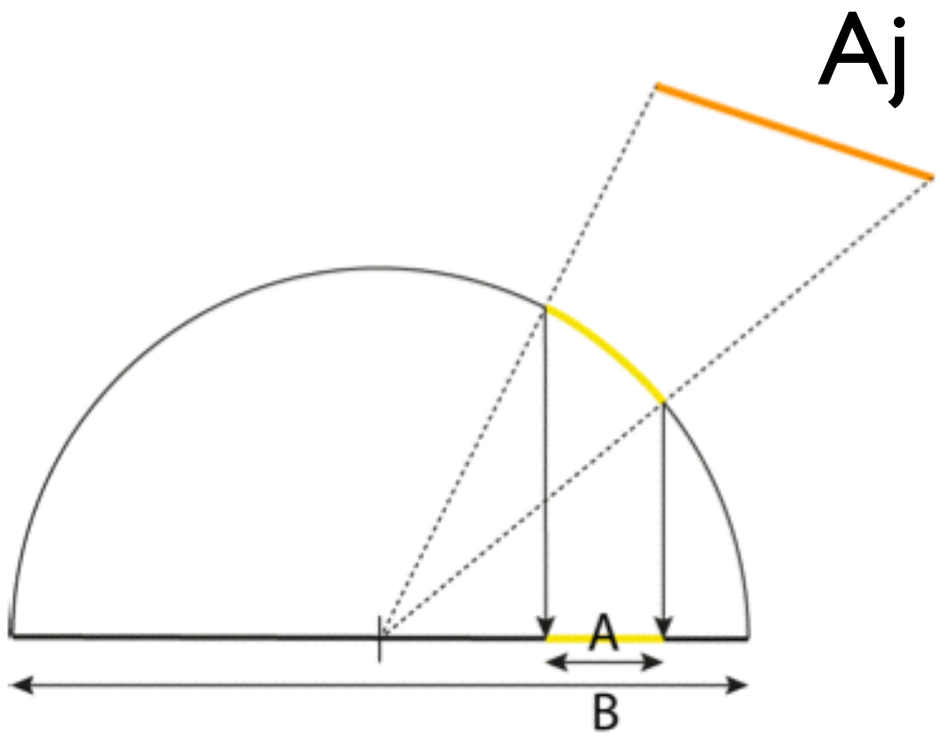


2. project the hemisphere orthographically on a unit circle.



3. divide by the area of the circle

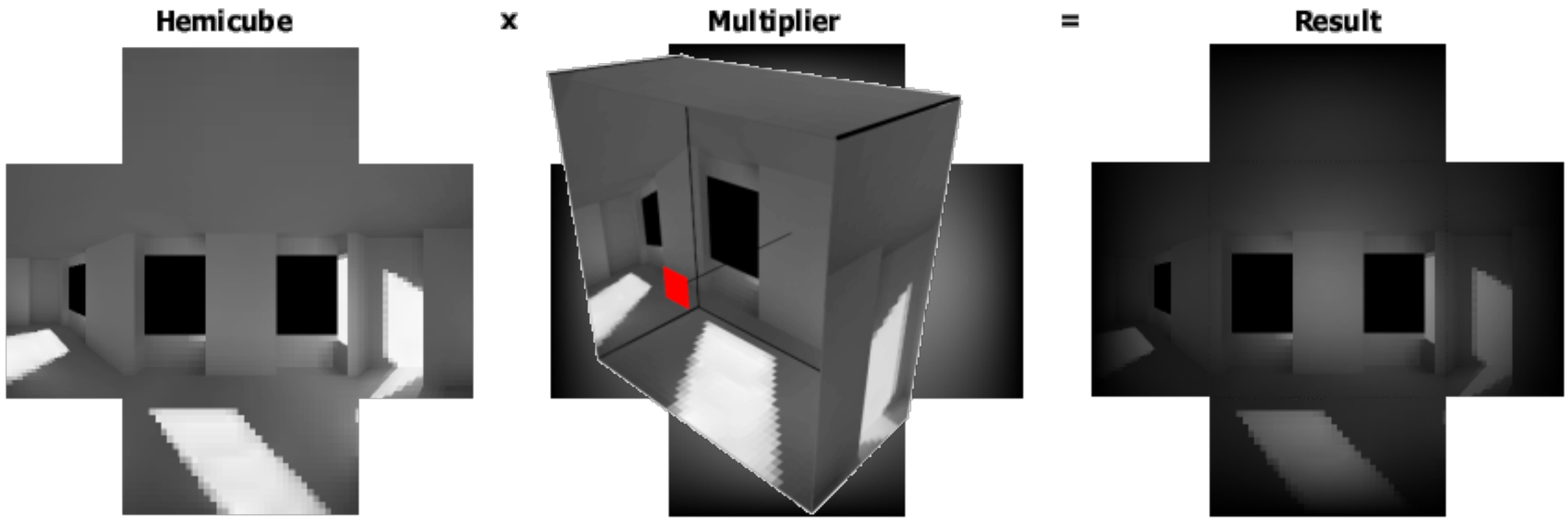
Nusselt Analog



Fij ≈ A/B

The hemicube method

A simpler method is to render the scene onto a hemicube and weight the pixels to account for the distortion.



Solving

The system of equations can be expressed as a matrix equation:

$$\begin{pmatrix} 1-\rho_1 F_{11} & -\rho_1 F1 & \cdots & -\rho_1 F_{1n} \\ -rho_2 F_{21} & 1-\rho_2 F_{22} & \cdots & -\rho_2 F_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ -rho_2 F_{n1} & -rho_2 F_{n2} & \cdots & 1-\rho_n F_{nn} \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{pmatrix}$$

In practice n is very large making exact solutions impossible.

Iterative approximation

One simple solution is merely to update the radiosity values in multiple passes:

$$B_i = E_i + \rho_i \sum_j B_j F_{ij}$$

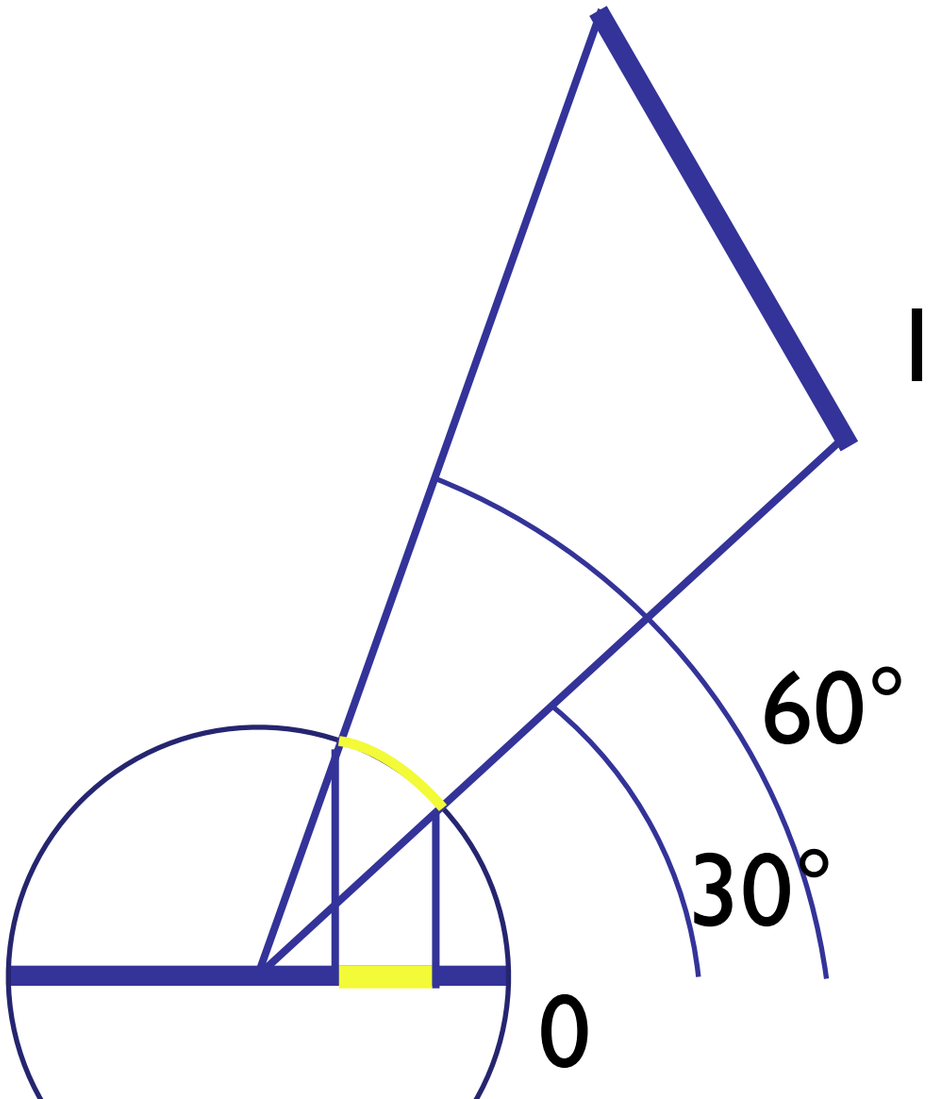
```
for each iteration:
  for each patch i:
    Bnew[i] = E[i]
    for each patch j:
      Bnew[i] +=
        rho[i] * F[i,j] * Bold[j];
  swap Bold and Bnew
```

$$\rho = 0.5$$

$$E[0] = 0$$

$$E[1] = 0.8$$

$$\cos(30) - \cos(60) \approx 0.37$$



F	0	1
0	0	0.185
1	0.185	0

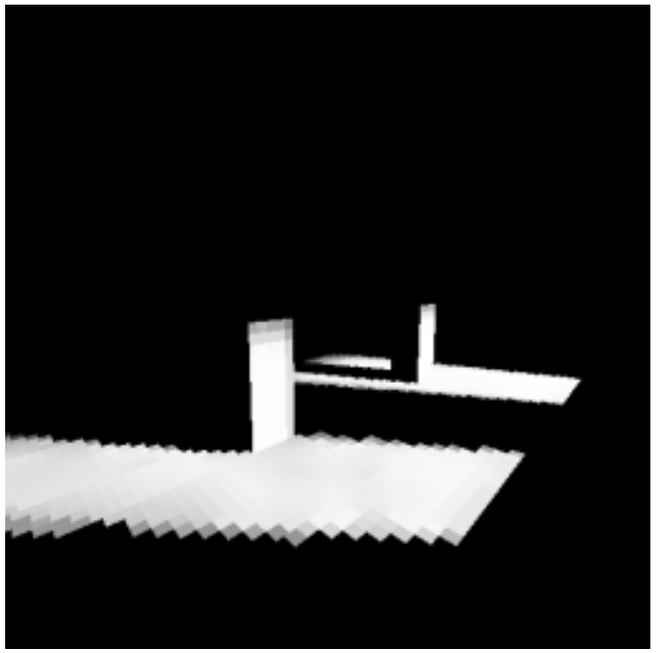
	B	
	0	1
1	0	0.8
2	0.072	0.8
3	0.074	0.807
4	0.075	0.807

Iterative approximation

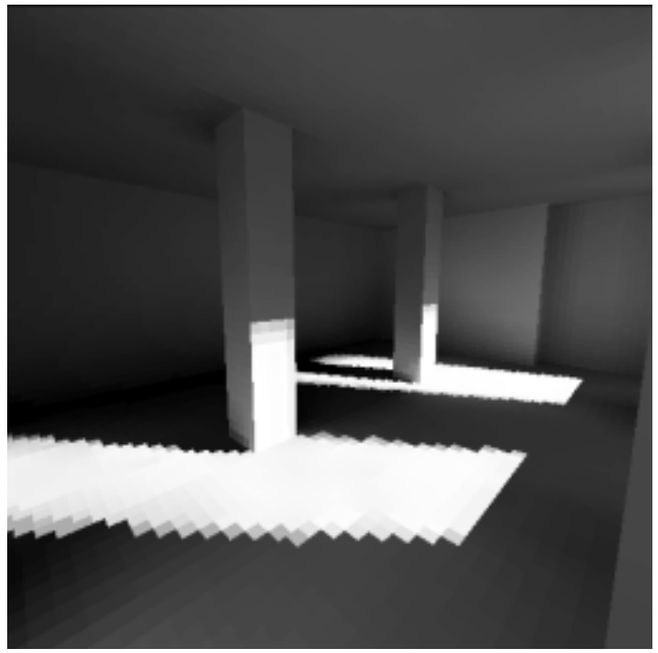
Using direct rendering

```
for each iteration:
   for each patch i:
     Bnew[i] = E[i]
     S = RenderScene(i,Bold)
     B = Sum of pixels in S
     Bnew[i] += rho[i]*B
swap Bold and Bnew
```

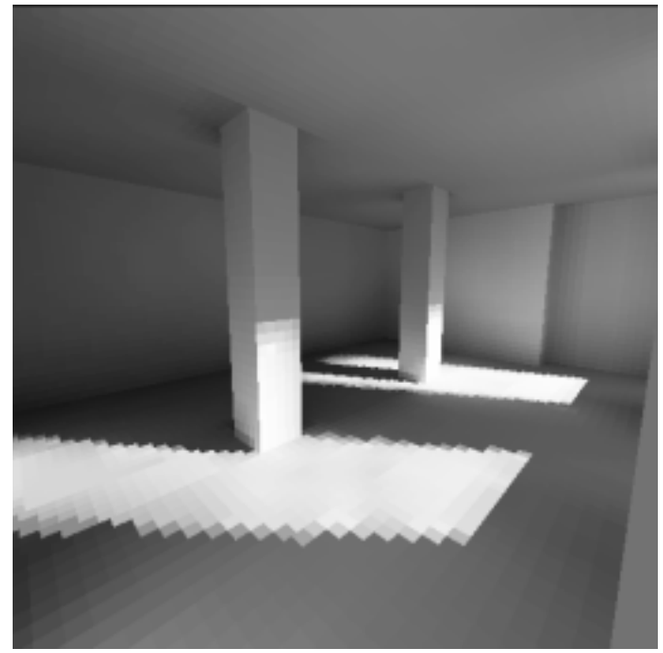
Iterative approximation



first pass
(direct lighting)

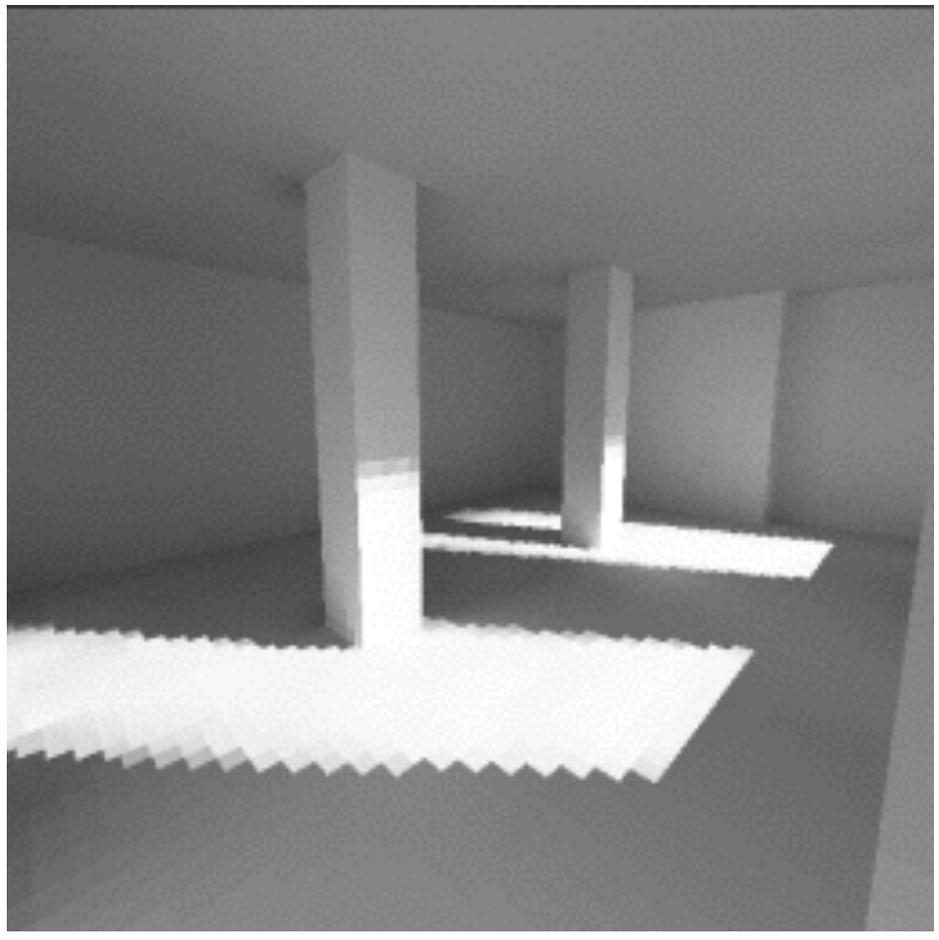


second pass
(one bounce)



third pass
(two bounces)

16th Pass



Progressive refinement

The iterative approach is inefficient as it spends a lot of time computing inputs from patches that make minimal or no contribution.

A better approach is to prioritise patches by how much light they output, as these patches will have the greatest contribution to the scene.

Progressive refinement

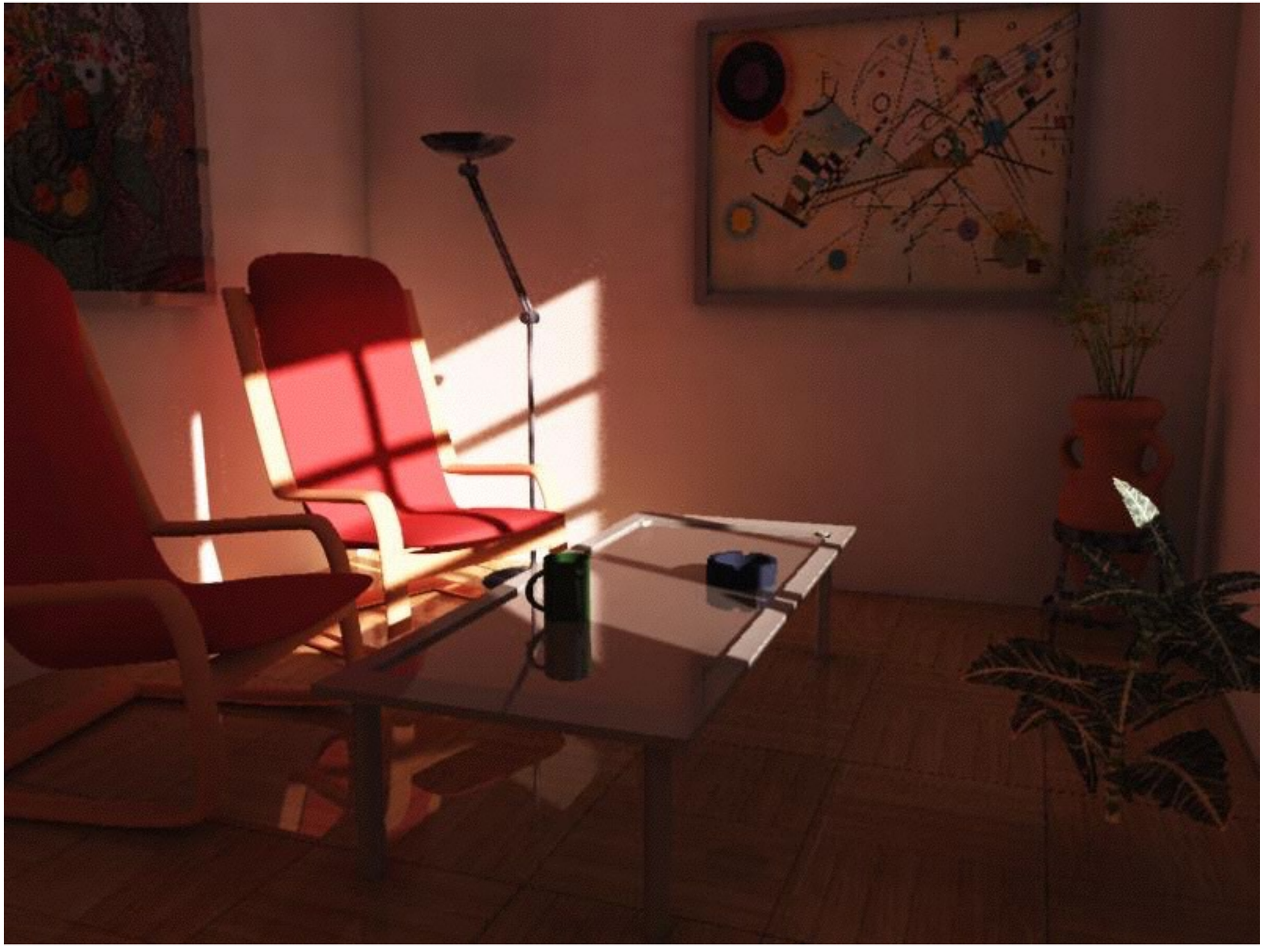
```
for each patch i:
    B[i] = dB[i] = E[i]
iterate:
  select patch i with max dB[i]:
  calculate F[i][j] for all j
  for each patch j:
    dRad = rho[j] * B[i] *
           F[i][j] * A[j] / A[i]
    B[j] += dRad
    dB[j] += dRad
  dB[i] = 0
```

In practice

Radiosity is computationally expensive, so rarely suitable for real-time rendering.

However, it can be used in conjunction with light mapping.

The payoff



Geometric light sources

Sources

http://freespace.virgin.net/hugo.elias/radiosity/radiosity.htm

http://www.cs.uu.nl/docs/vakken/gr/2011/gr_lectures.html

http://www.siggraph.org/education/materials/HyperGraph/radiosity/overview_2.htm

http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter39.html

Real-time Global Illumination

http://www.youtube.com/watch?v=Pq39Xb7OdH8

COMP3421

B-Splines

Quick Recap: Curves

We want a general purpose solution for drawing curved lines and surfaces. It should:

- Be easy and intuitive to draw curves
- Support a wide variety of shapes, including both standard circles, ellipses, etc and "freehand" curves.
- Be computationally cheap.

Bézier curves

Have the general form:

$$P(t) = \sum_{k=0}^{m} B_k^m(t) P_k$$

where m is the degree of the curve and $P_0 ... P_m$ are the control points.

Bernstein polynomials

$$B_k^m(t) = \binom{m}{k} t^k (1-t)^{m-k}$$

where:

$$\binom{m}{k} = \frac{m!}{k!(m-k)!}$$

is the binomial function.

Bernstein polynomials

$$P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(t-1)P_2 + t^3 P_3$$

For the most common case, m = 3:

$$\begin{aligned} B_0^3(t) &= (1-t)^3 \\ B_1^3(t) &= 3t(1-t)^2 \\ B_2^3(t) &= 3t^2(1-t) \\ B_3^3(t) &= t^3 \end{aligned}$$

Problems

Local control - Moving one control point affects the entire curve.

Incomplete - No circles, elipses, conic sections, etc.

Problem: Local control

These curves suffer from non-local control.

Moving one control point affects the entire curve.

Each Bernstein polynomial is active (non-zero) over the entire interval [0,1]. The curve is a blend of these functions so every control point has an effect on the curve for all t from [0,1]

Splines

A spline is a smooth piecewise-polynomial function (for some measurement of smoothness).

The places where the polynomials join are called knots.

A joined sequence of Bézier curves is an example of a spline.

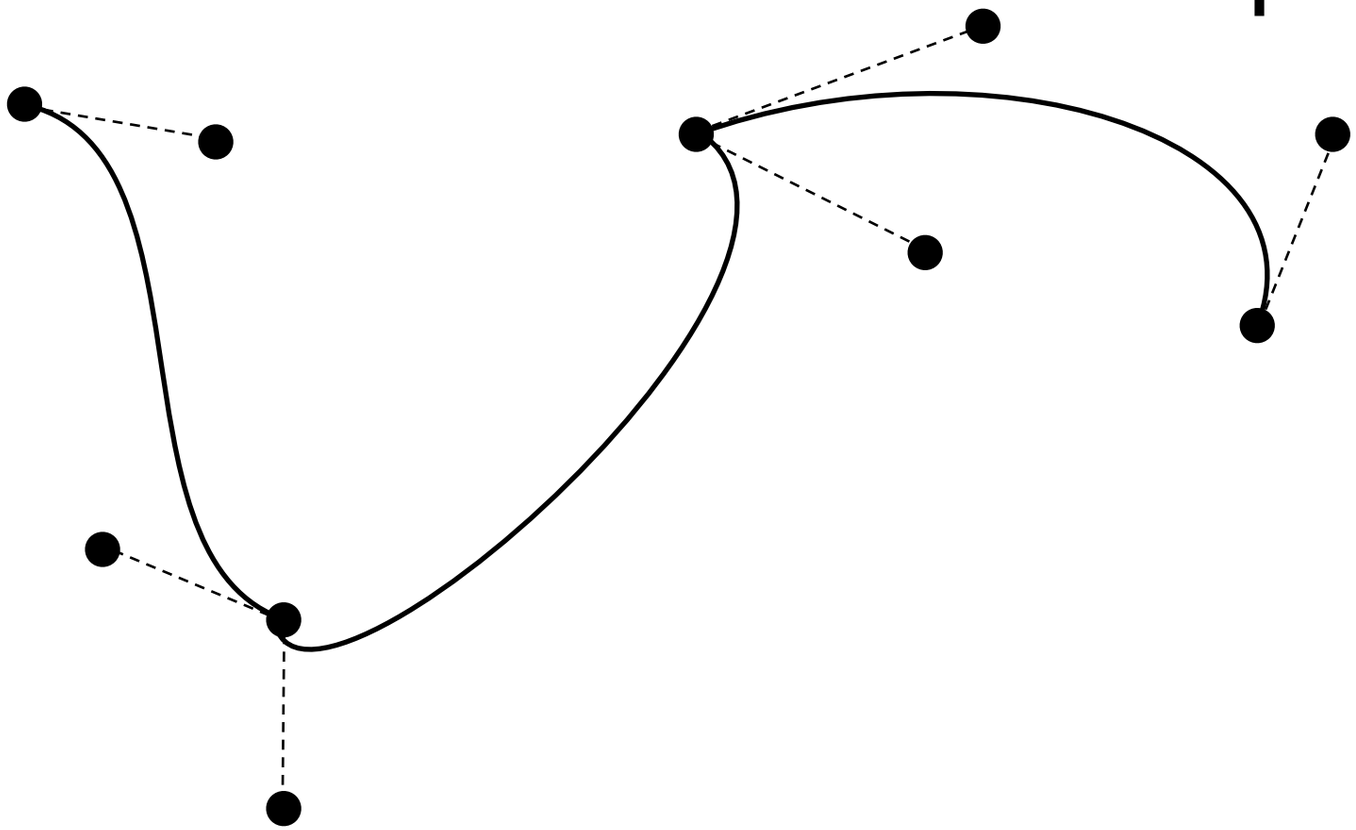
Local control

A spline provides local control.

A control point only affects the curve within a limited neighbourhood.

Bézier splines

We can draw longer curves as sequences of Bézier sections with common endpoints:



Parametric Continuity

A curve is said to have C^n continuity if the *n*th derivative is continuous for all t:

$$\mathbf{v}_n(t) = \frac{d^n P(t)}{dt^n}$$

C^0: the curve is connected.

C^1: a point travelling along the curve doesn't have any instantaneous changes in velocity.

C2: no instantaneous changes in acceleration

Geometric Continuity

A curve is said to have G^n continuity if the normalised derivative is continuous for all t.

$$\hat{\mathbf{v}}_n(t) = \frac{\mathbf{v}_n(t)}{|\mathbf{v}_n(t)|}$$

G^1 means tangents to the curve are continuous

G^2 means the curve has continuous curvature.

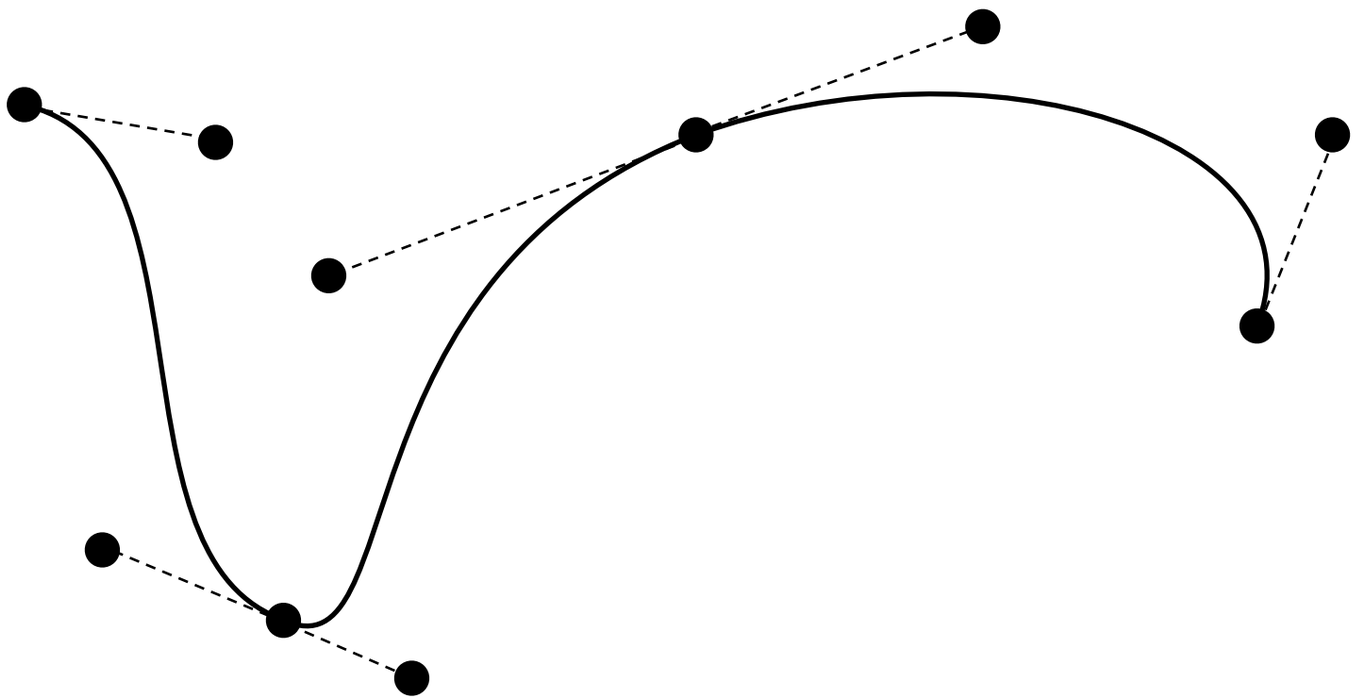
Continuity

Geometric continuity is important if we are drawing a curve.

Parametric continuity is important if we are using a curve as a guide for motion.

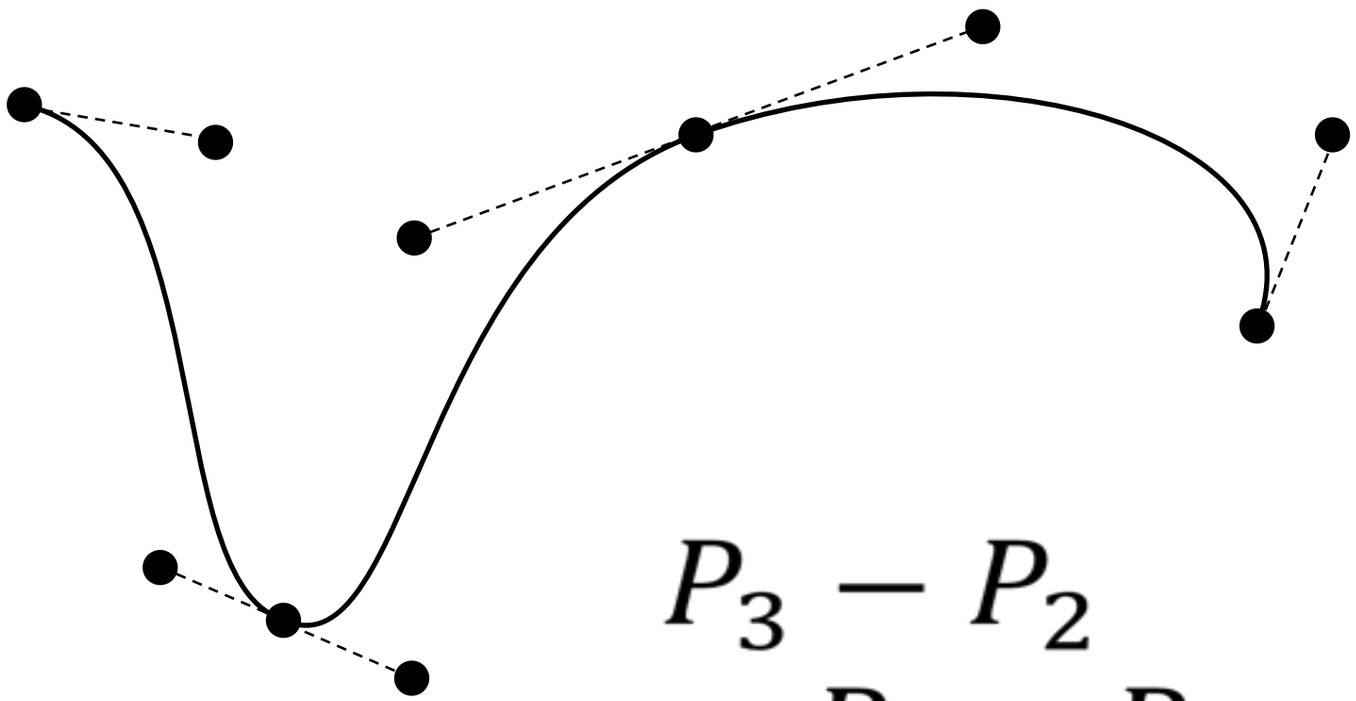
Bézier splines

If the control points are collinear, the the curve has G^1 continuity:



Bézier splines

If the control points are collinear and equally spaced, the curve has C^1 continuity:



$$P_3 - P_2 = P_4 - P_3$$

B-splines

We can generalise Bézier splines into a larger class called basis splines or B-splines.

A B-spline of degree m has equation:

$$P(t) = \sum_{k=0}^{L} N_k^m(t) P_k$$

where L is the number of control points, with

$$L > m$$

B-splines

The $N_k^m(t)$ function is defined recursively:

$$
\begin{aligned}
N_k^m(t) &= \left(\frac{t - t_k}{t_{m+k} - t_k}\right) N_k^{m-1}(t) \\
&\quad + \left(\frac{t_{m+k+1} - t}{t_{m+k+1} - t_{k+1}}\right) N_{k+1}^{m-1}(t) \\
N_k^0(t) &= \begin{cases} 1 & \text{if } t_k < t \leq t_{k+1} \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

(Note: this formulation differs slightly from the one in the textbook)

Knot vector

The sequence $(t_0, t_1, \ldots, t_{m+L})$ is called the knot vector.

The knots are ordered so $t_k \leq t_{k+1}$

Knots mark the limits of the influence of each control point.

Control point P_k affects the curve between knots t_k and t_{k+m+1}.

Number of Knots

The number of knots in the knot vector is always equal to the number of control points plus the order of the curve. E.g., a cubic (m=3) with five control points has 9 items in the knot vector. For example:

(0,0.125,0.25,0.375,0.5,0.625,0.75,0.875,1)

Uniform / Non-uniform

Uniform B-splines have equally spaced knots.

Non-uniform B-splines allow knots to be positioned arbitrarily and even repeat.

A multiple knot is a knot value that is repeated several times.

Multiple knots create discontinuities in the derivatives.

Continuity

A polynomial of degree m has C^m continuity.

A knot of multiplicity k reduces the continuity by k.

So, a uniform B-spline of degree m has C^{m-1} continuity.

Interpolation

A uniform B-spline approximates all of its control points.

A common modification is to have knots of multiplicity $m+1$ at the beginning and end in order to interpolate the endpoints. This is called clamping.

Moving Controls and Knots

Moving Controls: Adjacent control points on top of one another causes the curve to pass closer to that point. With m adjacent control points the curve passes through that point.

Moving Knots: Across a normal knot the continuity for and degree curve is C^{m-1}. Each extra knot with the same value reduces continuity at that value by one.

Quadratic and Cubic

The most commonly used B-splines are quadratic (m=2) and cubic (m=3).

Uniform quadratic splines have C^1 (and G^1) continuity.

Uniform cubic splines have C^2 (and G^2) continuity.

Bezier and B-Spline

A Bézier curve of degree m is a clamped uniform B-spline of degree m with L=m+1 control points.

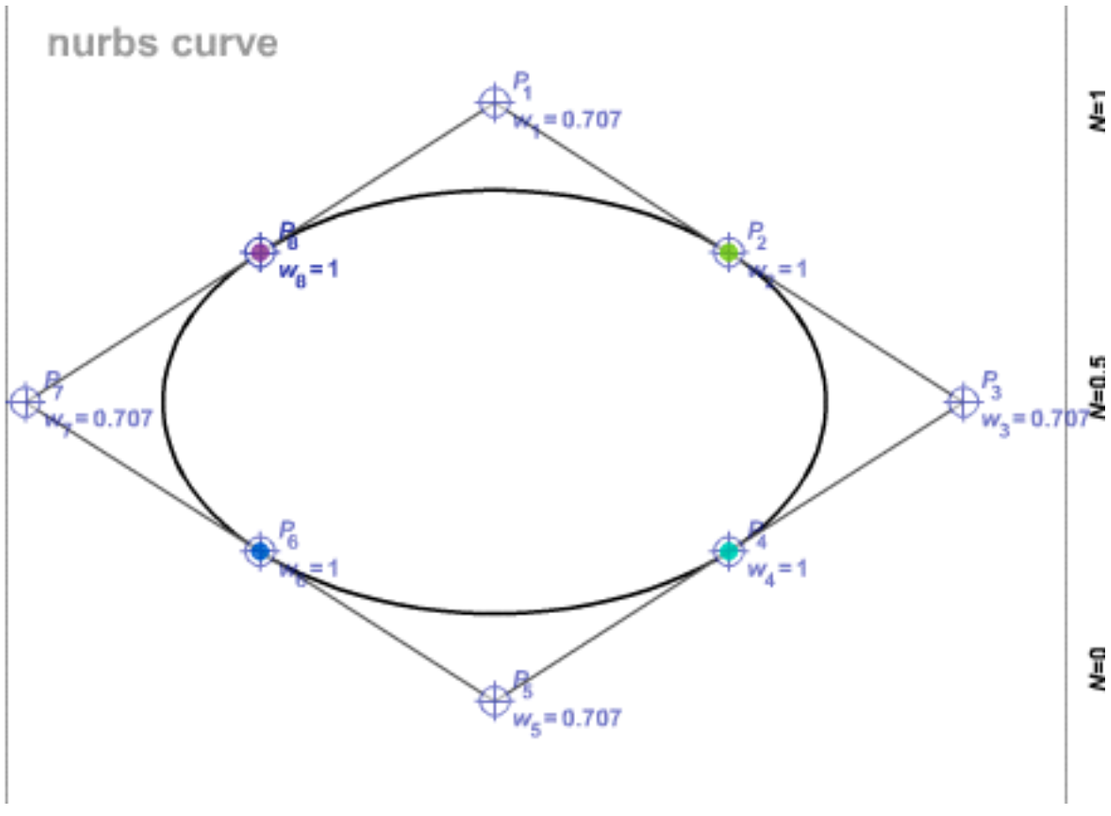
A Bézier spline of degree m is a sequence of bezier curves connected at knots of multiplicity m.

A quadratic piecewise Bézier knot vector with seven control points

will look like this [0 0 0 1 1 2 2 3 3 3].

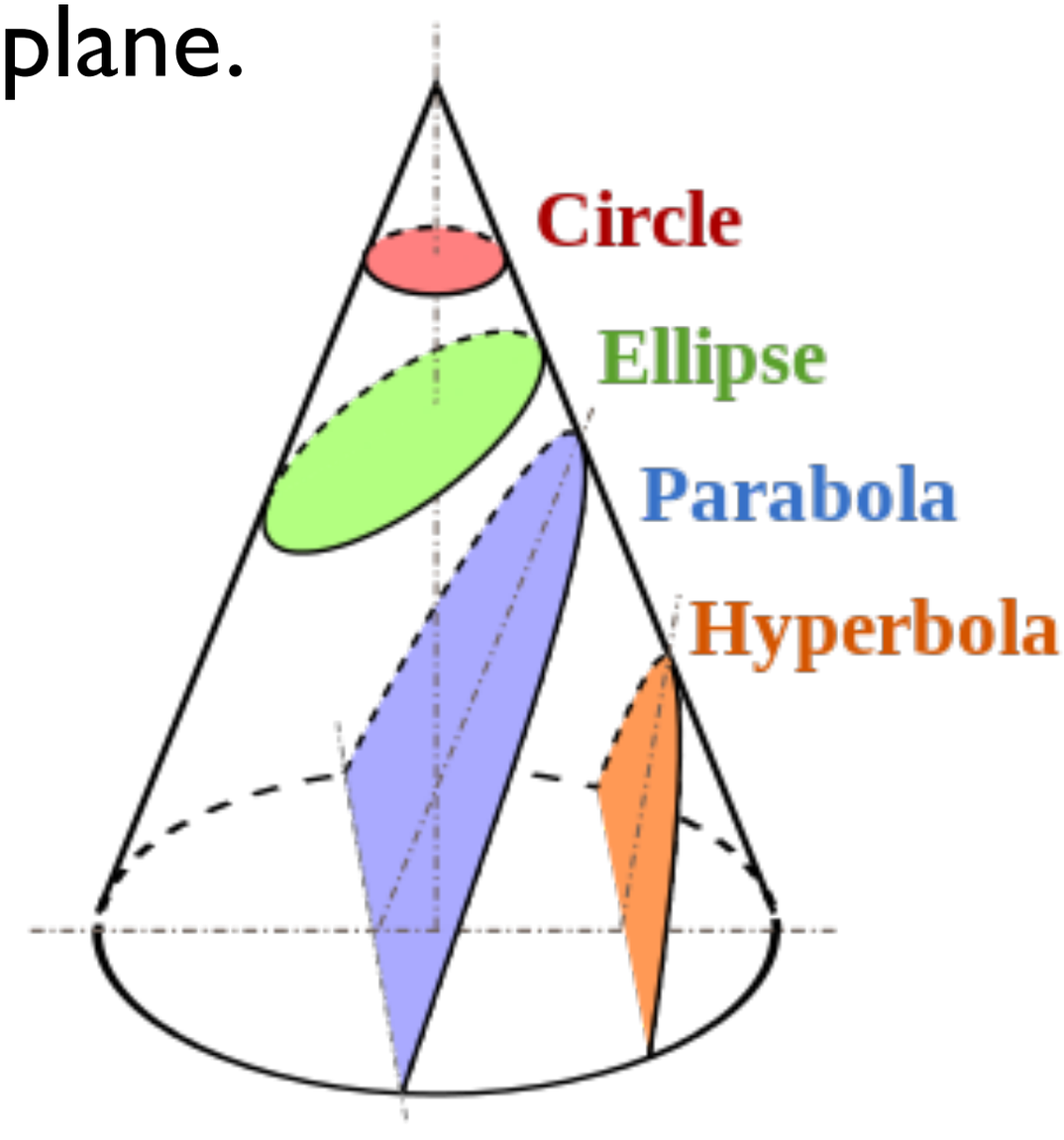
Stop. Demo Time.

http://geometrie.foretnik.net/files/NURBS-en.swf



Incomplete

Conic sections are what happens when a cone intersects a plane.



Rational Bézier Curves

We can create a greater variety of curve shapes if we weight the control points:

$$P(t) = \frac{\sum_{k=0}^{m} w_k B_k^m(t) P_k}{\sum_{k=0}^{m} w_k B_k^m(t)}$$

A higher weight draws the curve closer to that point.

This is called a rational Bézier curve.

Rational Bézier Curves

Rational Bézier curves can exactly represent all conic sections (circles, ellipses, parabolas, hyperbolas).

This is not possible with normal Bézier curves.

If all weights are the same, it is the same as a Bezier curve

Rational B-splines

We can also weight control points in B-splines to get rational B-splines:

$$P(t) = \frac{\sum_{k=0}^{L} w_k N_k^m(t) P_k}{\sum_{k=0}^{L} w_k N_k^m(t)}$$

NURBS

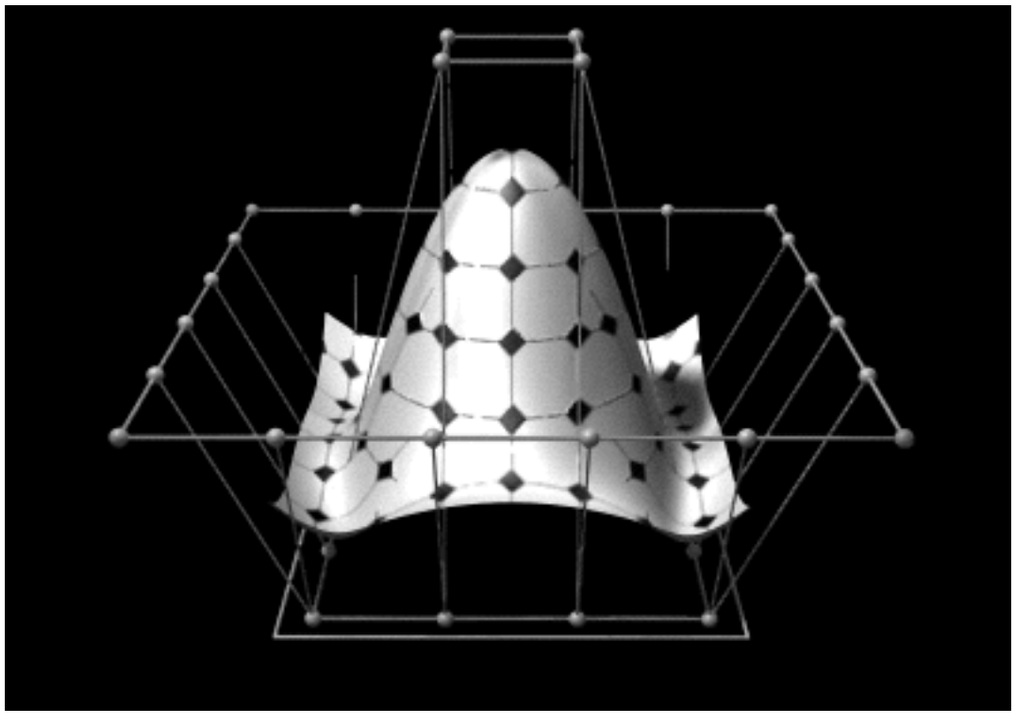
Non-uniform rational B-splines are known as NURBS.

NURBS provide a power yet efficient and designer-friendly class of curves.

Closed curves

A unclamped uniform B-spline of degree m is a closed loop if the first m control points match the last m control points.

Surfaces



Surfaces

We can create 2D surfaces by parameterising over two variables:

$$P(s,t) = \sum_{i=0}^{L} \sum_{j=0}^{M} F_i(s) F_j(t) P_{i,j}$$

Where $F_k(t)$ is any particular spline function we choose (Bezier, B-spline, NURBS)

and $P_{i,j}$ denote an LxM array of control points.