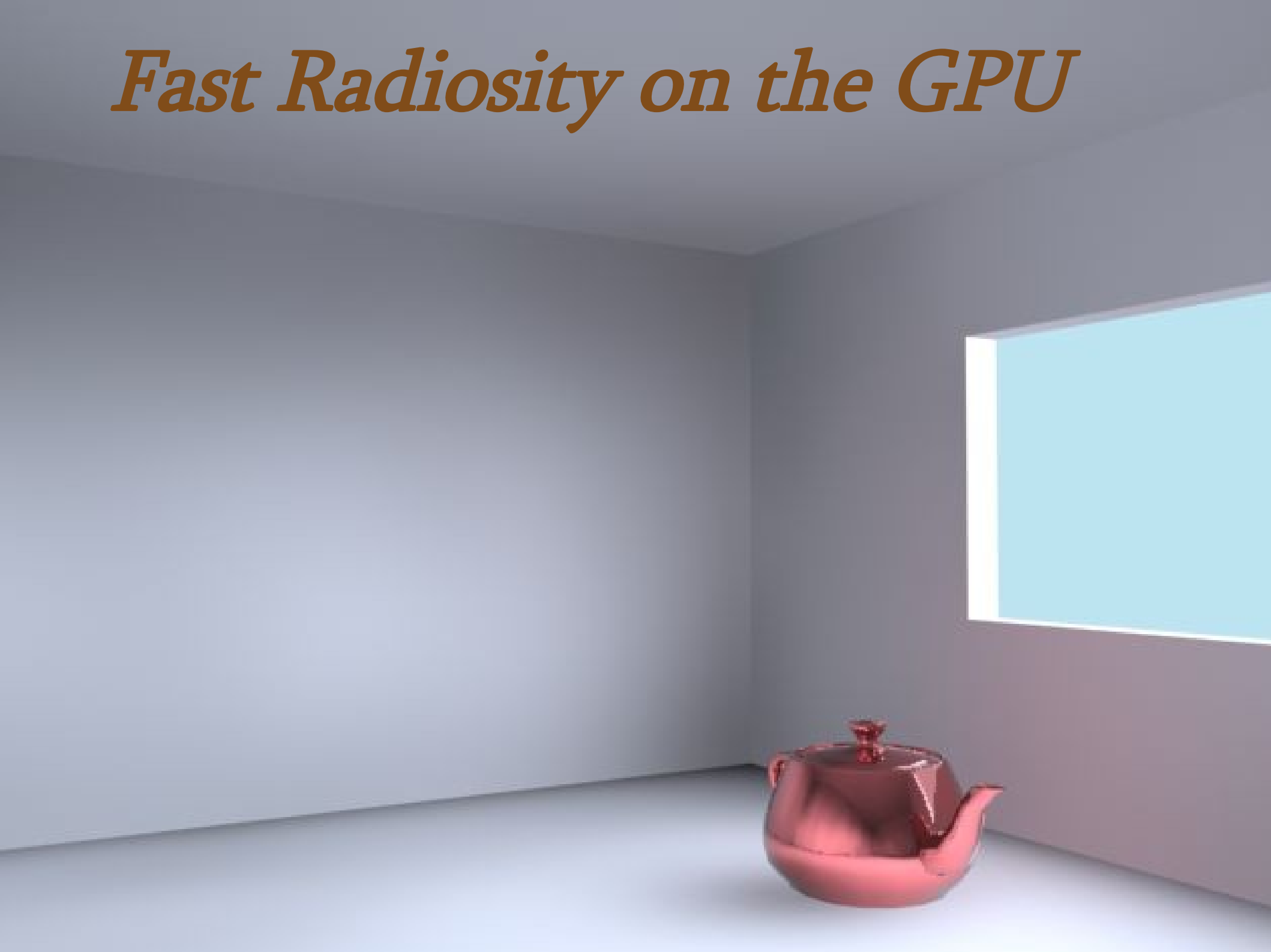
Fast Radiosity on the GPU

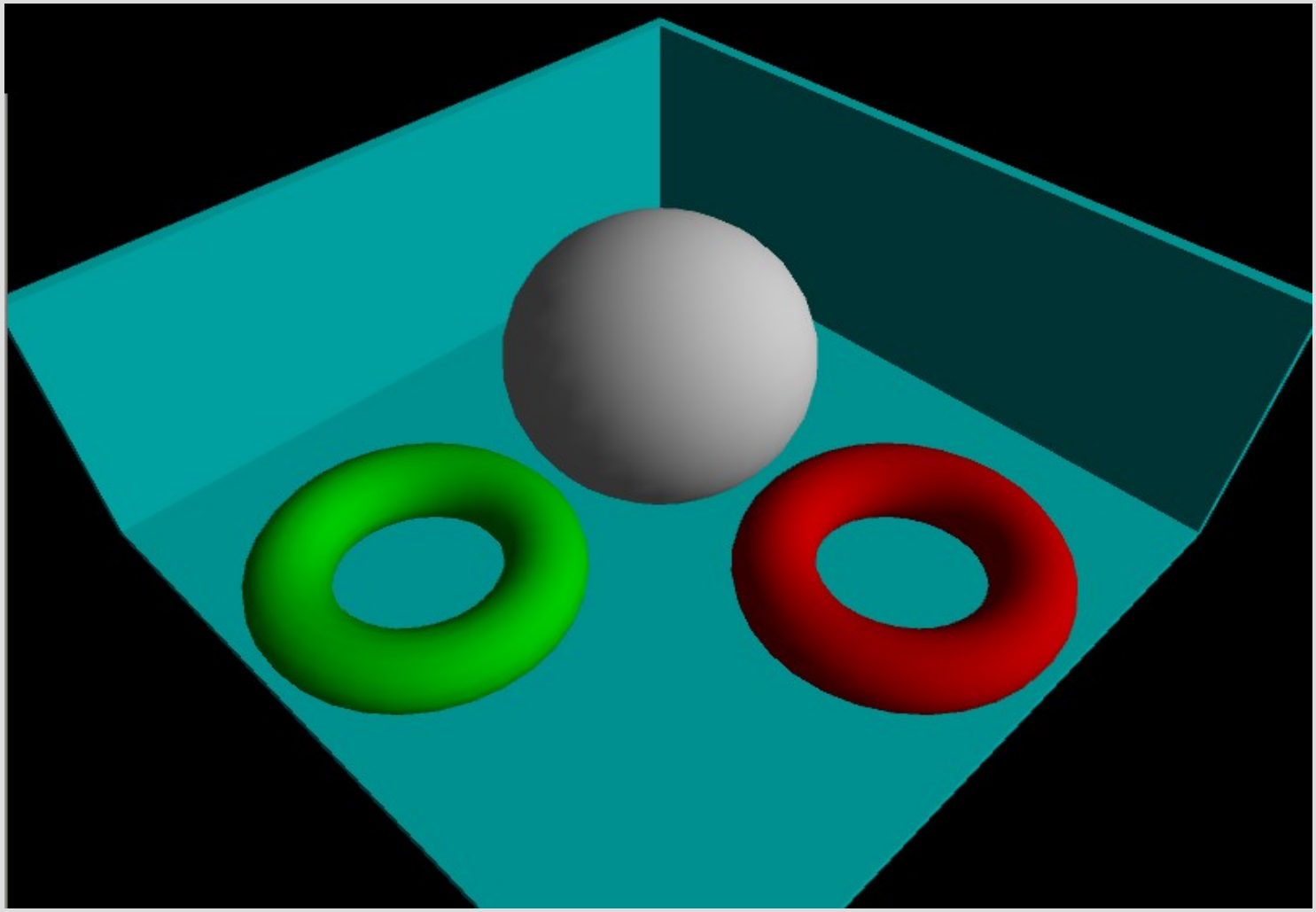


Fast Radiosity on the GPU

- Contenido:
  - Iluminación local e iluminación global.
  - Fundamentos de *Radiosity*.
  - Algoritmo clásico.
  - Aceleración por CPU.
  - Aceleración por GPU.
  - Formas de simular *Radiosity*.
  - Demostraciones.
  - Conclusiones.

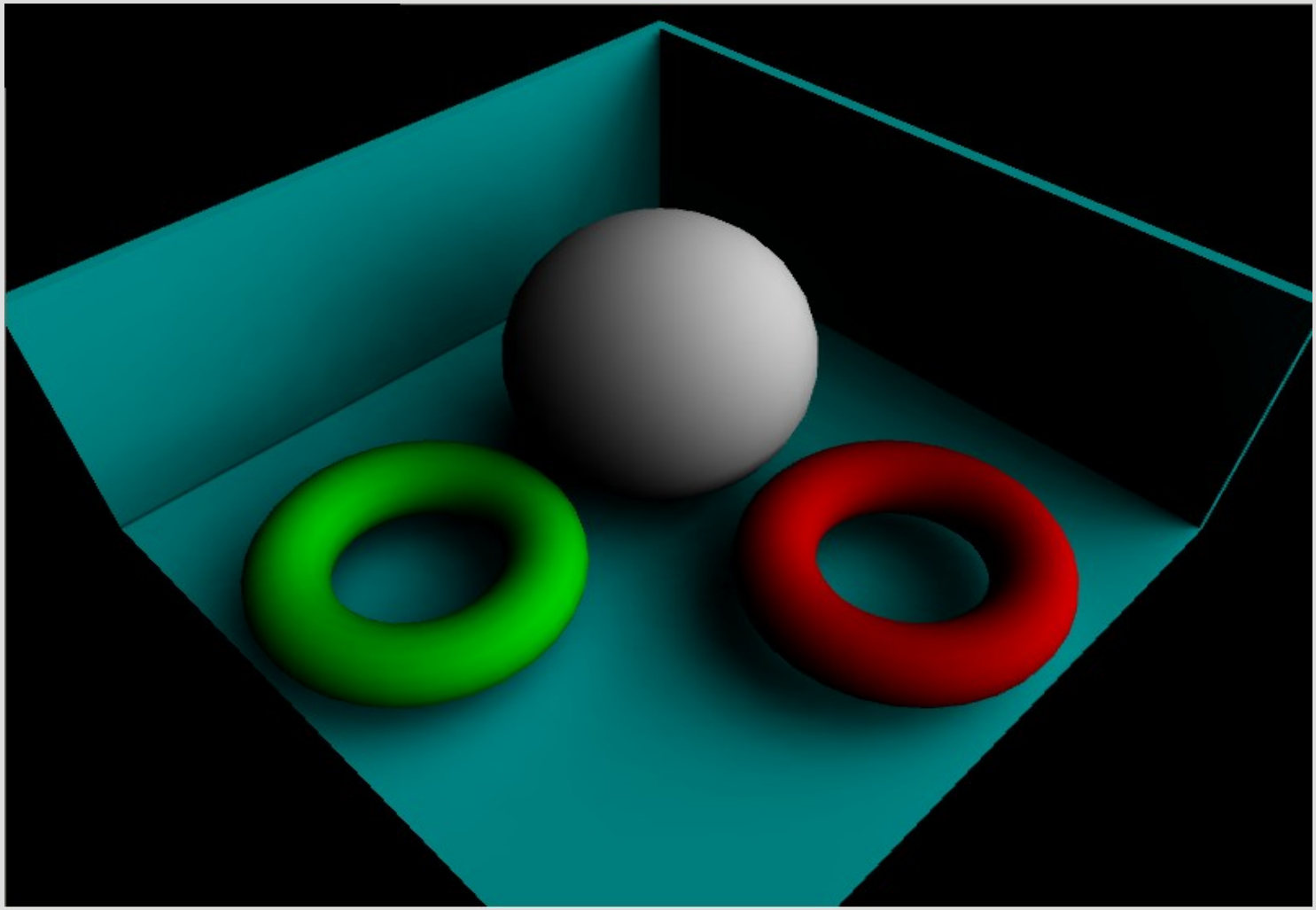
Iluminación Local

- **Iluminación local.**

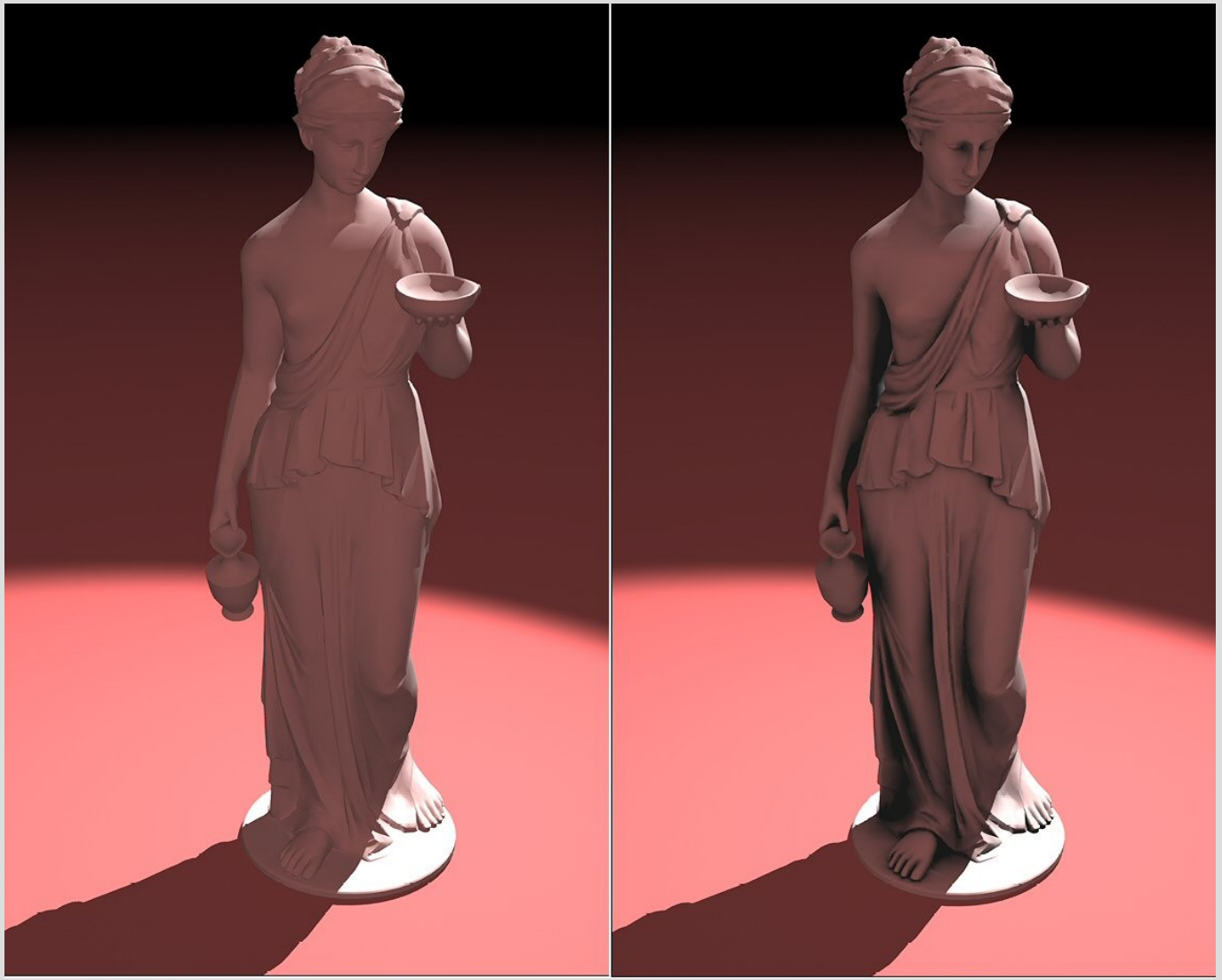


Iluminación Global

- **Iluminación global.**



Iluminación Global



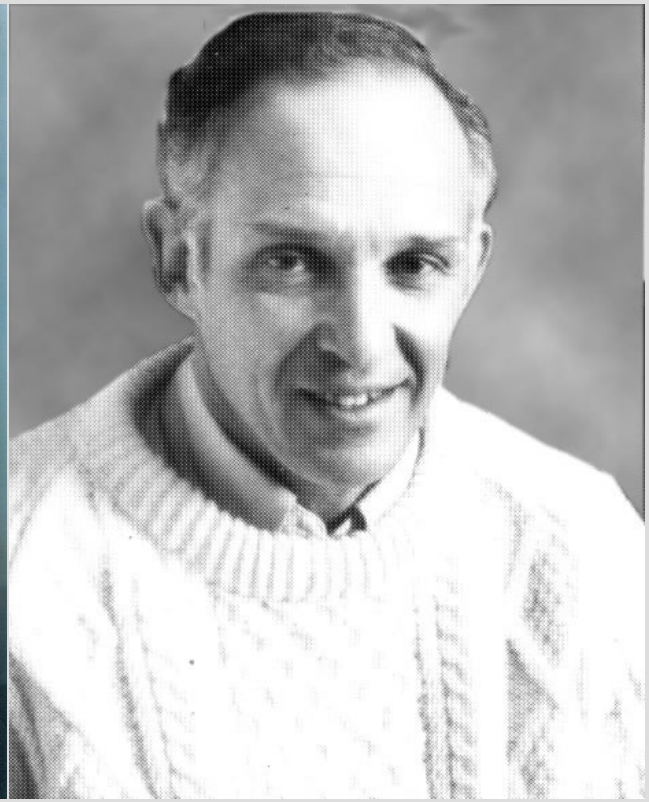
Iluminación Global


NU PAREE CENTRAL
WE WILL CLEANSE NU EARTH.

Fundamentos de *Radiosity*

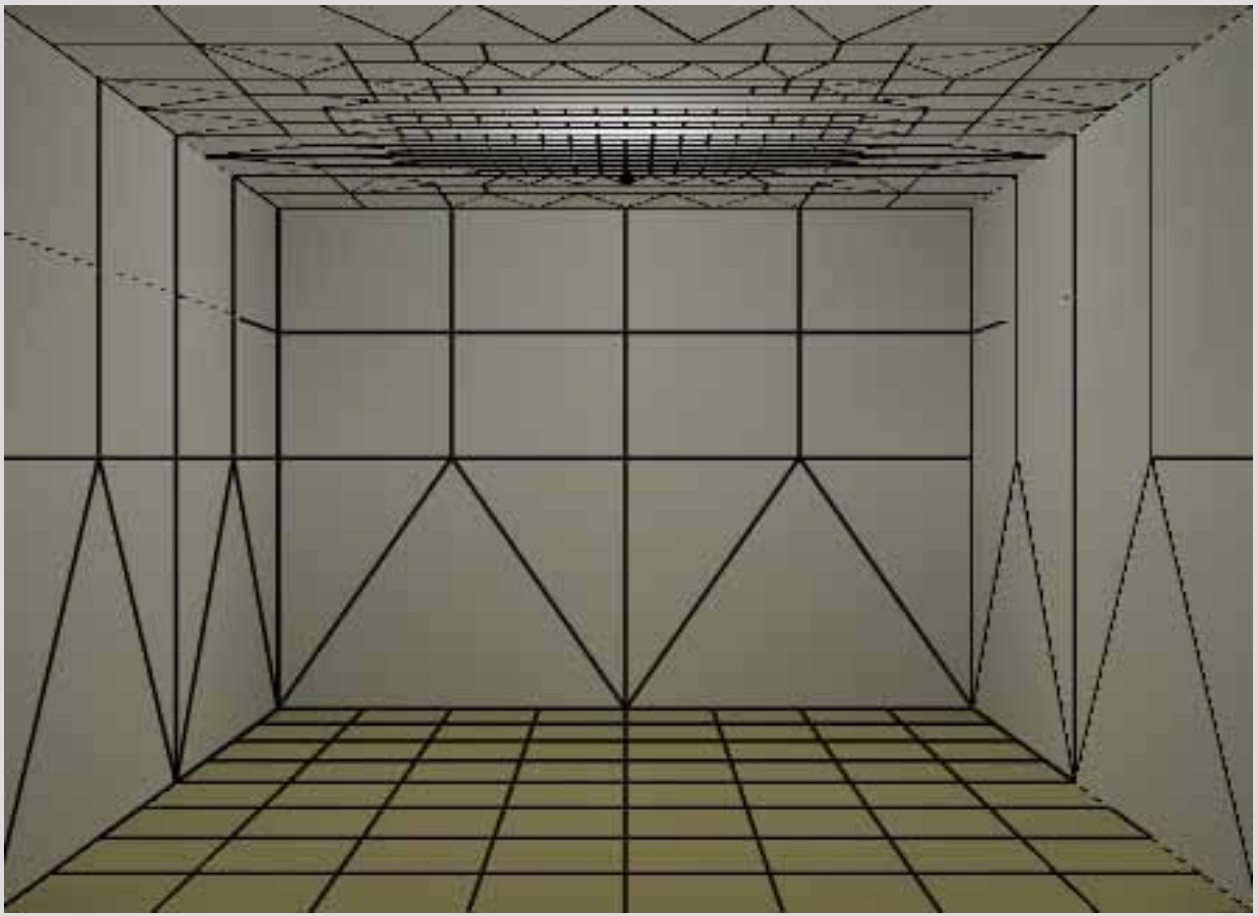
- ***Radiosity***:
  - Desarrollado en 1984 por Cindy Goral, Kenneth Torrance, Donald Greenberg y Bennett Battaile.





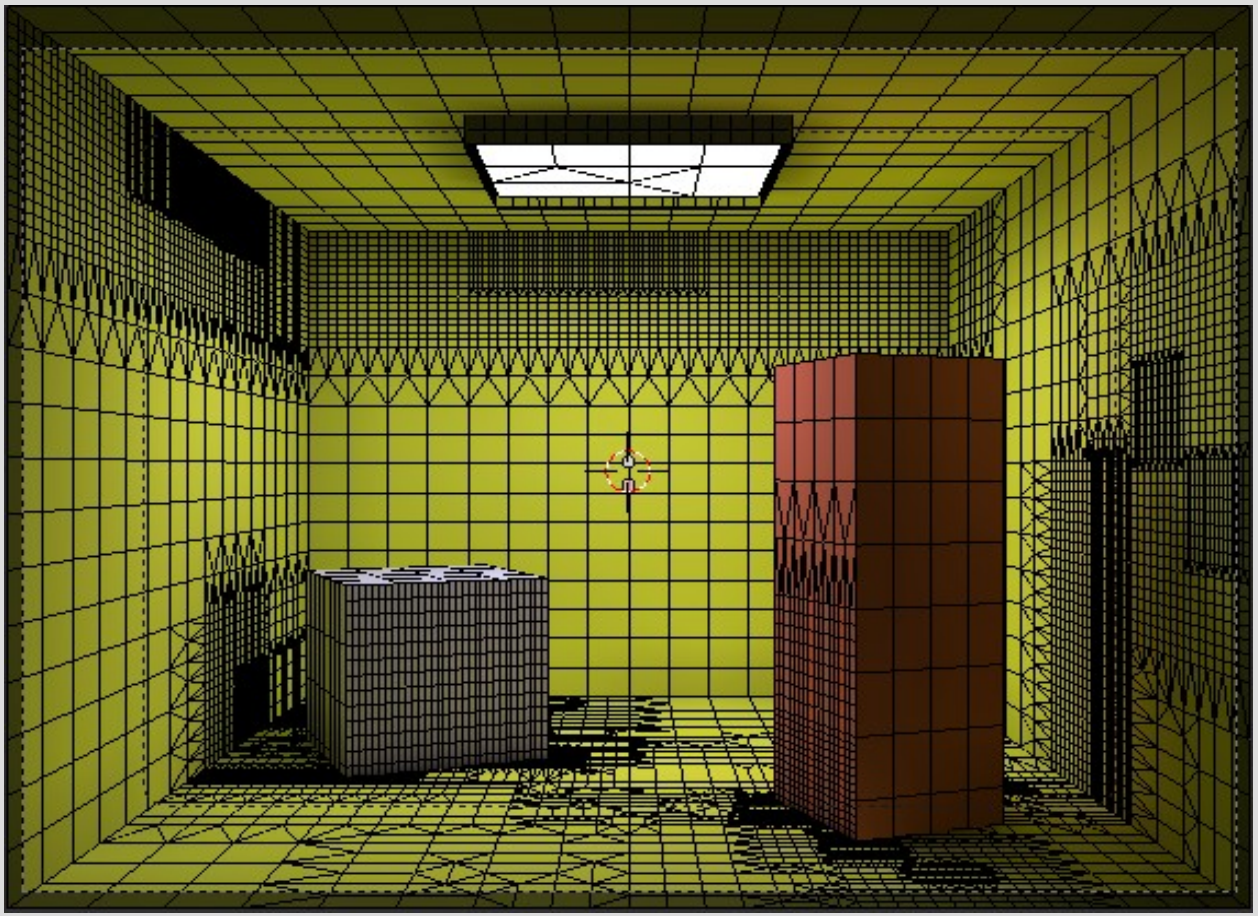
Fundamentos de *Radiosity*

- Conceptos:
  - Elemento.



Fundamentos de *Radiosity*

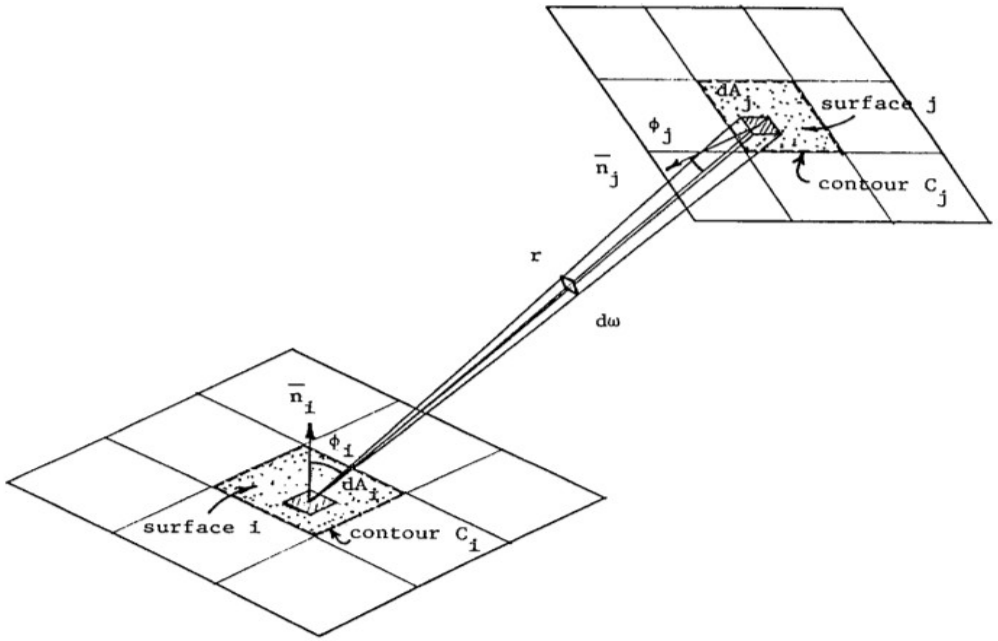
- Conceptos:
  - Parche.



Fundamentos de *Radiosity*

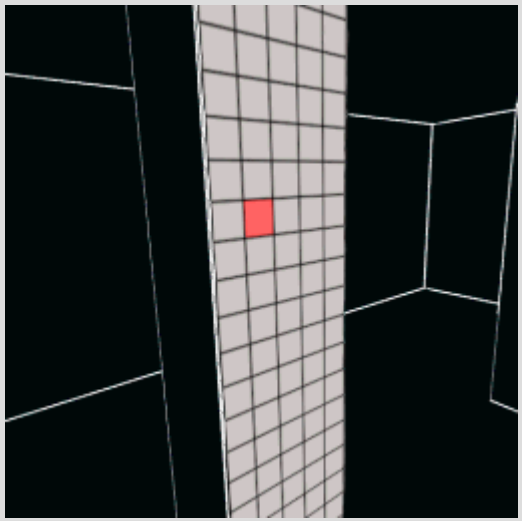
- Conceptos:
  - Factor de forma.

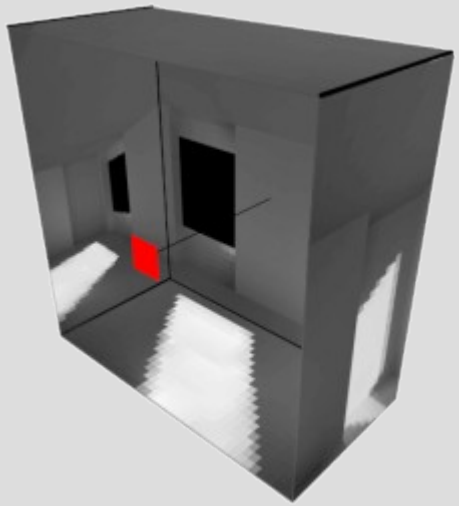
$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos(\phi_i)\cos(\phi_j)\, dA_i\, dA_j}{\pi\, r^2}$$

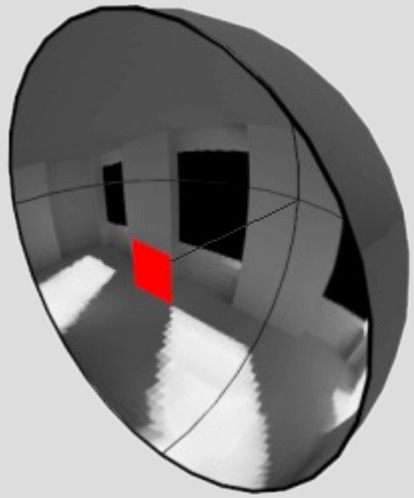


Fundamentos de *Radiosity*

- Conceptos:
  - Semicubo (o semiesfera).







Fundamentos de *Radiosity*

- Ecuación de *Radiosity*:

$$b_j = e_j + \rho_j \sum_{i=1}^{N} b_i F_{ij}; \; j = 1, \ldots, N$$

- b_j = Intensidad de la luz que parte del parche *j*.
- e_j = Coeficiente de emisión del parche *j*.
- p_j = Coeficiente de reflectancia del parche *j*.
- F_{ij} = Factor de forma entre los parches *j* e *i*.

Algoritmo clásico

- Algoritmo clásico:

Función *Radiosity*

CargarEscena();

SubdividirEscena();

CalcularFactoresDeForma();

ArmarSistemaDeEcuaciones();

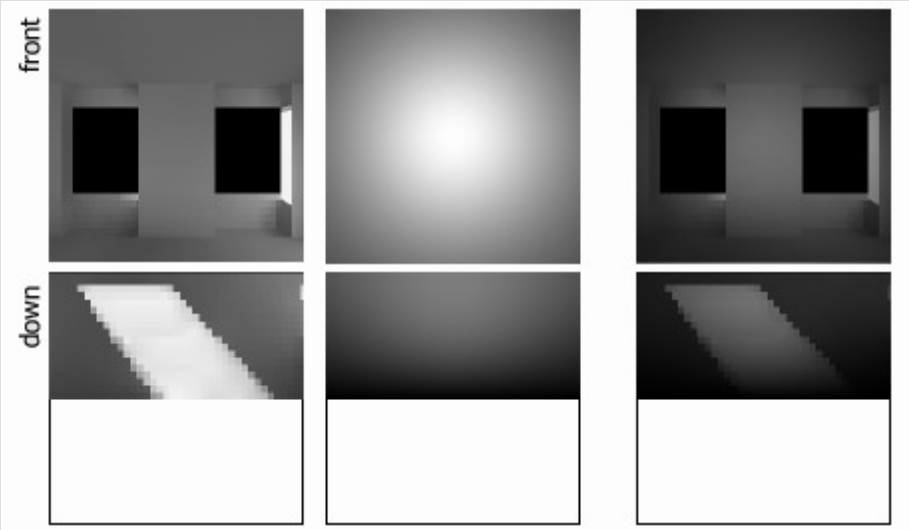
ResolverSistemaDeEcuaciones();

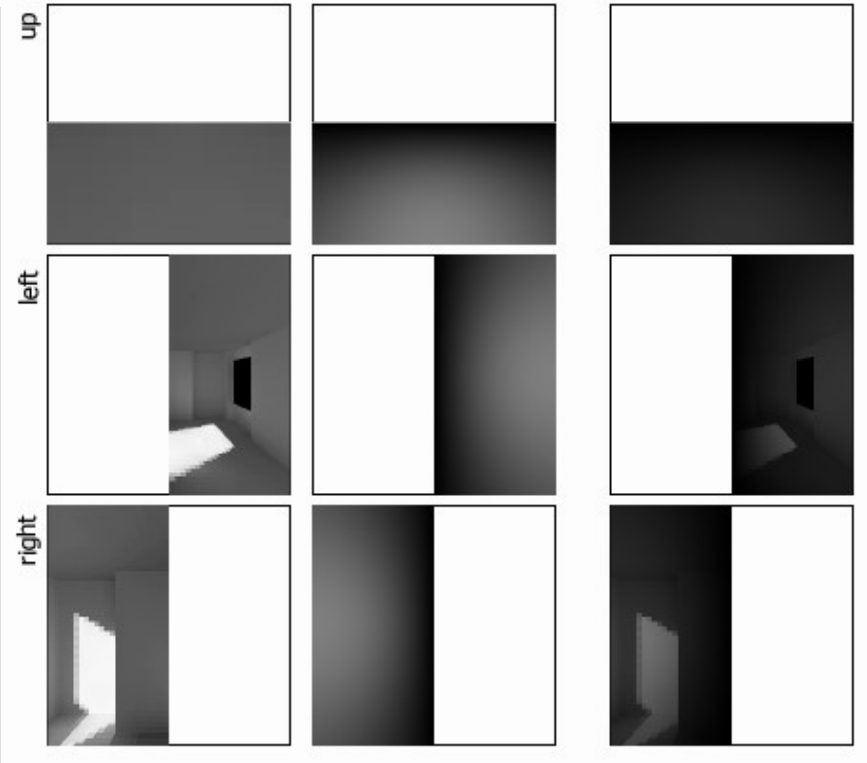
FinFunción *Radiosity*;

Algoritmo clásico

- Cálculo de los factores de forma:

$$F_{ij} = \frac{1}{2\pi A_i} \oint_{C_j} \oint_{C_i} \left[\ln(r)\, dx_i\, dx_j + \ln(r)\, dy_i\, dy_j + \ln(r)\, dz_i\, dz_j \right]$$





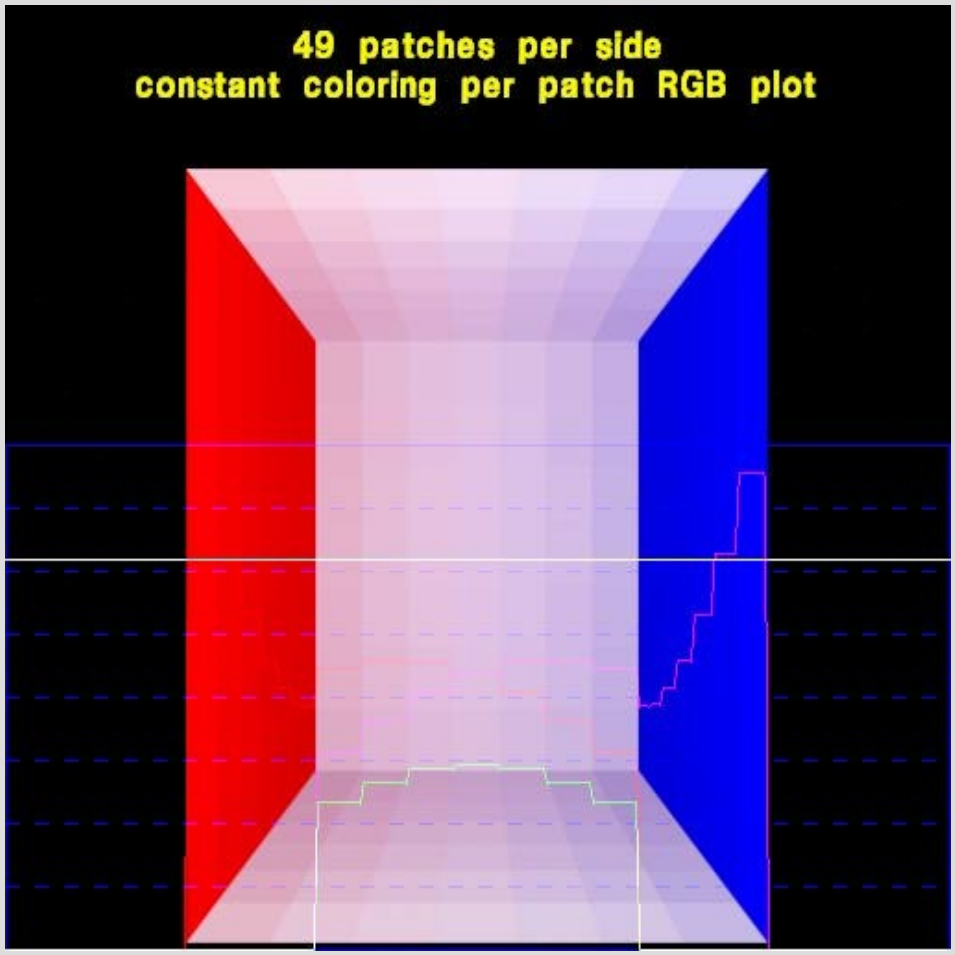
Algoritmo clásico

- Sistema de ecuaciones lineales:

$$\begin{bmatrix} 1-\rho_1 F_{1,1} & -\rho_1 F_{1,2} & \cdots & -\rho_1 F_{1,N} \\ -\rho_2 F_{2,1} & 1-\rho_2 F_{2,2} & \cdots & -\rho_2 F_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_N F_{N,1} & -\rho_N F_{N,2} & \cdots & 1-\rho_N F_{N,N} \end{bmatrix} x \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_N \end{bmatrix}$$

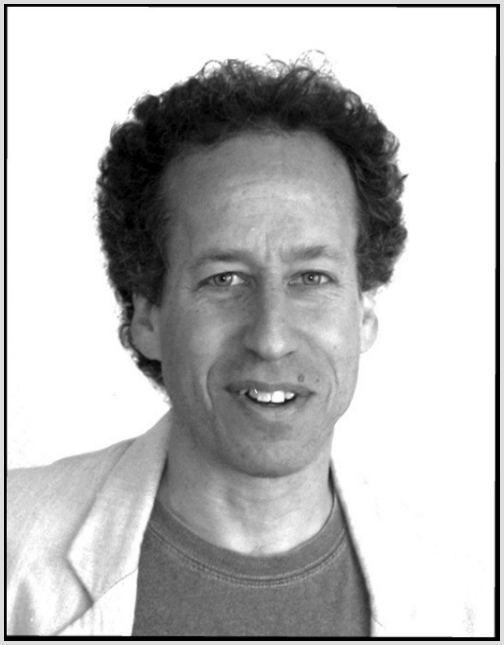
Algoritmo clásico

- Resultados:

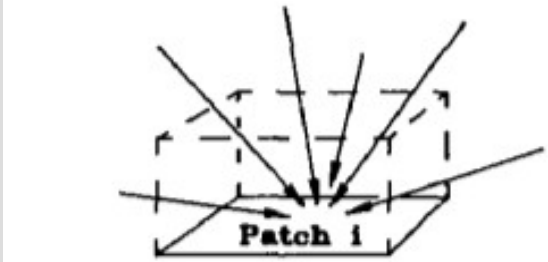


Aceleración por CPU

- **Refinamiento progresivo**.



Michael Cohen



GATHERING vs. SHOOTING

$$B_i = E_i + \sum_{j=1}^{N} (\rho_i F_{ij}) B_j$$

For all j:

$$B_j = B_j + B_i (\rho_j F_{ji})$$

where: $F_{ji} = F_{ij} A_i / A_j$

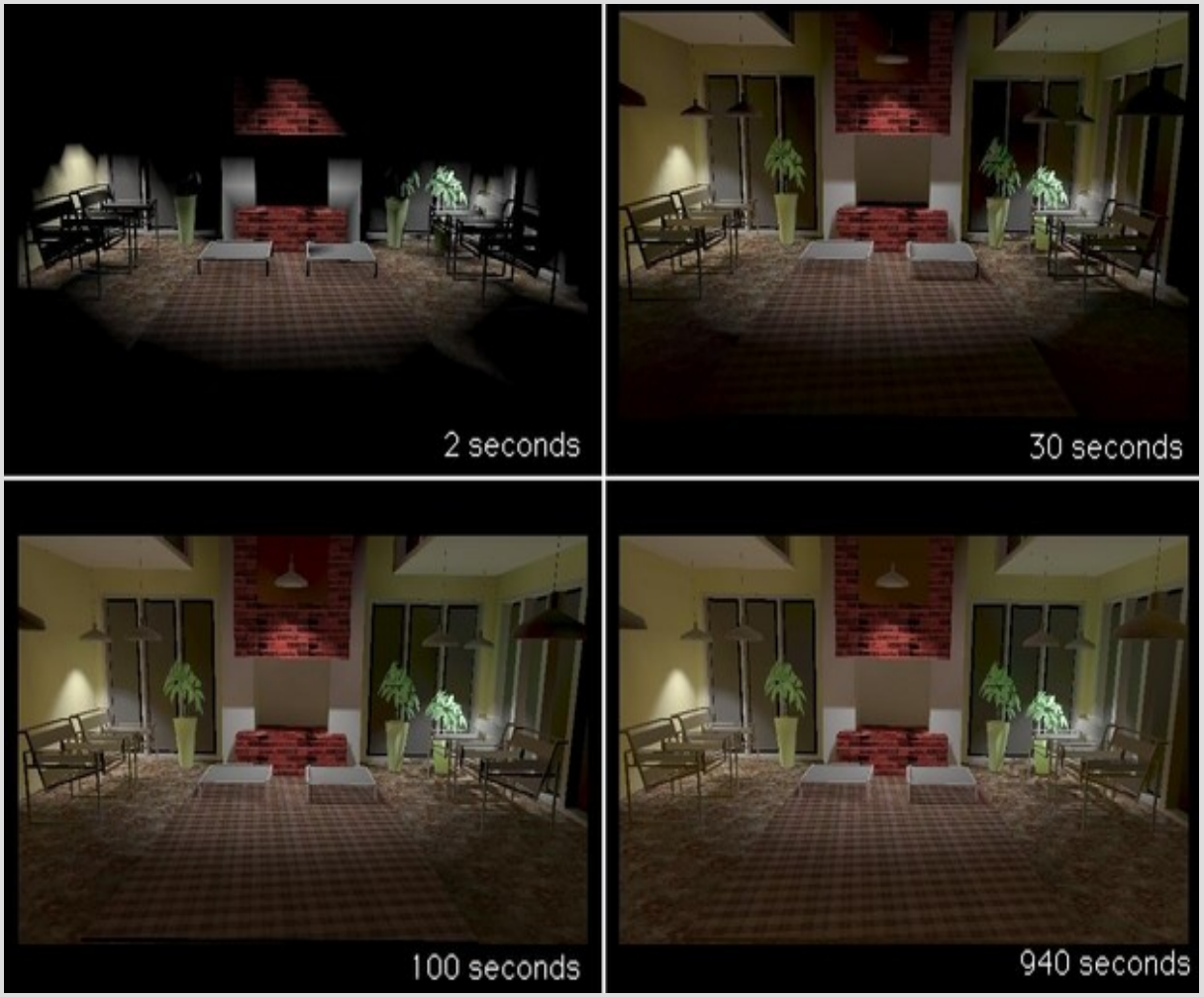
Aceleración por CPU

- Algoritmo:

```
función RefinamientoProgresivo
        CalcularTerminoAmbiental();
        InicializarDisparoEnEmisión();
        mientras (no convergencia) hacer
                SeleccionarEmisorActual();
                ProyectarSemiCubo();
                por cada (elemento) hacer
                        CalcularIncrementoRadiosidad();
                        SumarRadiosidadPonderada();
                fin por cada
                DeterminarCambioAmbiental();
                InterpolarColorVértices();
                Dibujar();
        fin mientras
fin función RefinamientoProgresivo
```

Aceleración por CPU

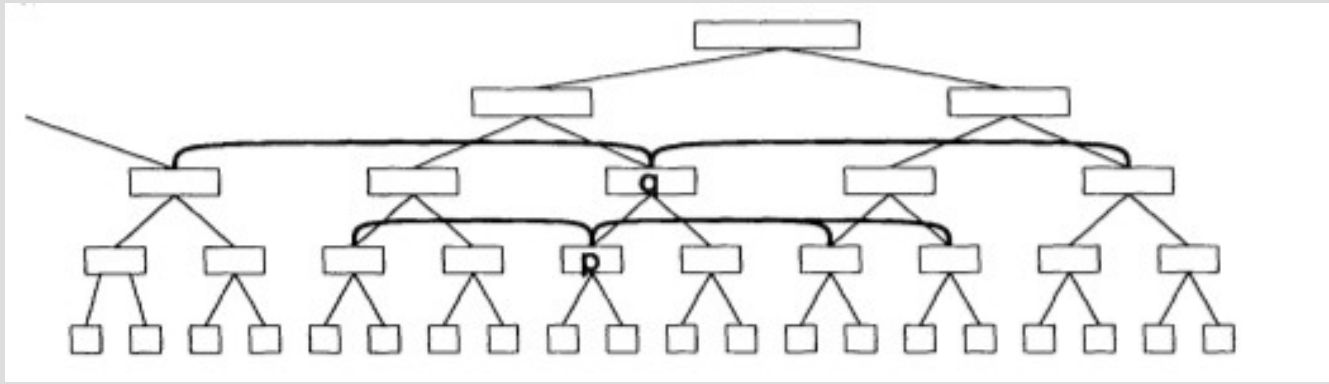
- Resultados:

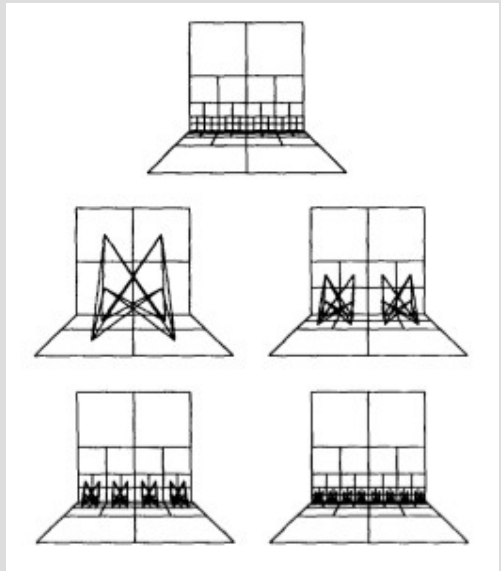


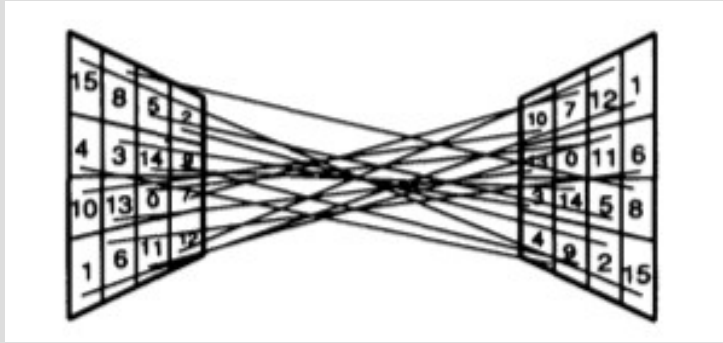
Aceleración por CPU

- **Subdivisión Adaptativa (Enfoque Jerárquico)**

Pat Hanrahan







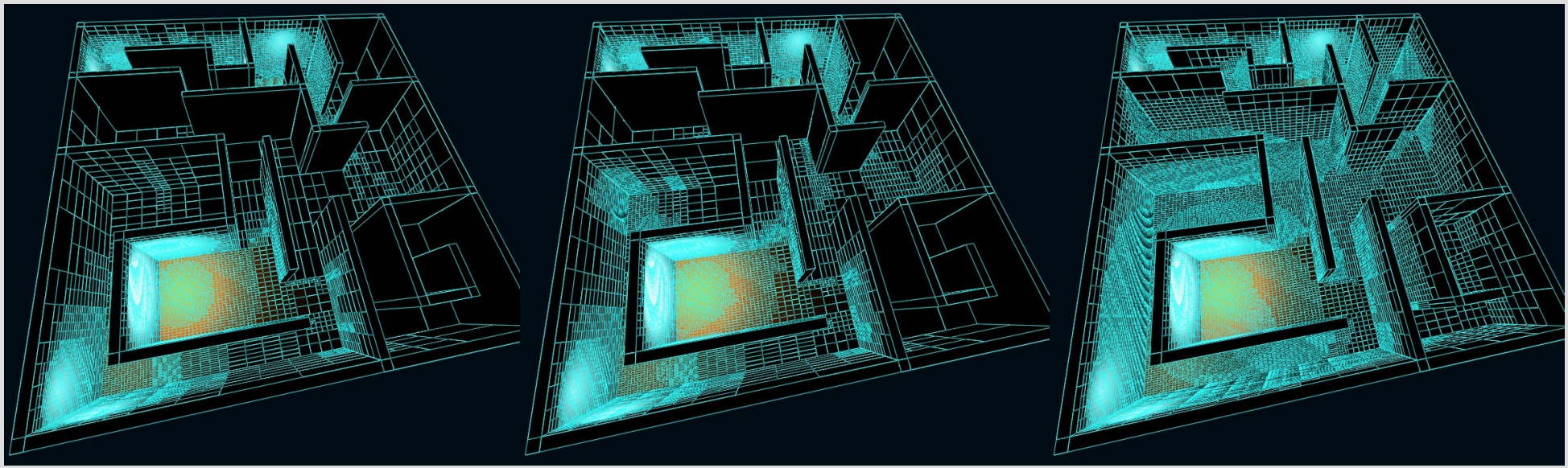
Aceleración por CPU

- Algoritmo:

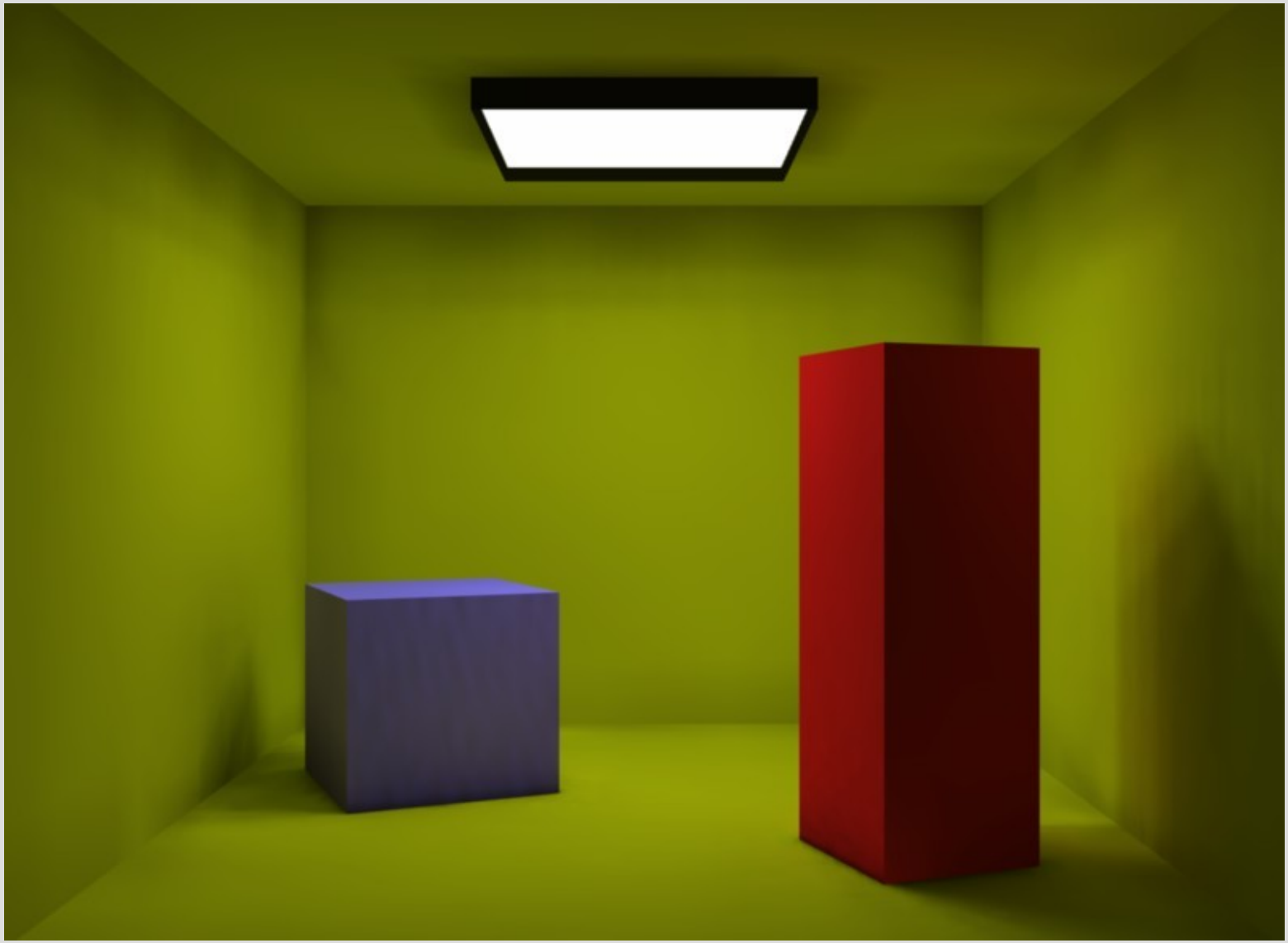
```
función Refinar (p, q, Feps, Aeps)
        Fpq = estimarFactorForma(p,q);
        Fqp = estimarFactorForma(q,p);
        si ( Fpq < Feps y Fqp < Feps)
                Enlazar(p,q);
        sino
                si ( Fpq >Fqp)
                        si( Subdividir( q, Aeps) )
                                Refinar (p, q->ne, Feps, Aeps);
                                Refinar (p, q->nw, Feps, Aeps);
                                Refinar (p, q->se, Feps, Aeps);
                                Refinar (p, q->sw, Feps, Aeps);
                        sino Enlazar(p,q); finsi
                sino
                        si( Subdividir( p, Aeps) )
                                Refinar (q, p->ne, Feps, Aeps);
                                Refinar (q, p->nw, Feps, Aeps);
                                Refinar (q, p->se, Feps, Aeps);
                                Refinar (q, p->sw, Feps, Aeps);
                        sino Enalazar(p,q); finsi
                finsi
        finsi
finfunción Refinar
```

Aceleración por CPU

- Subdivisión adaptativa:



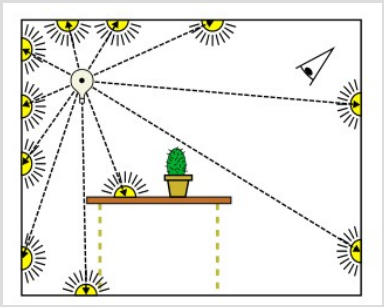
Aceleración por CPU

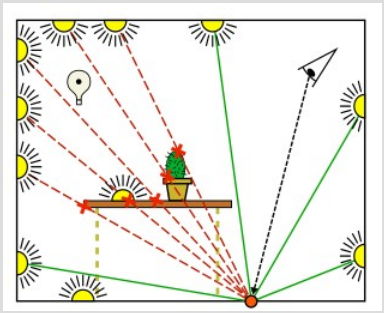


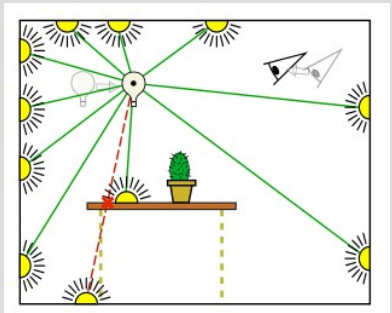
Aceleración por GPU

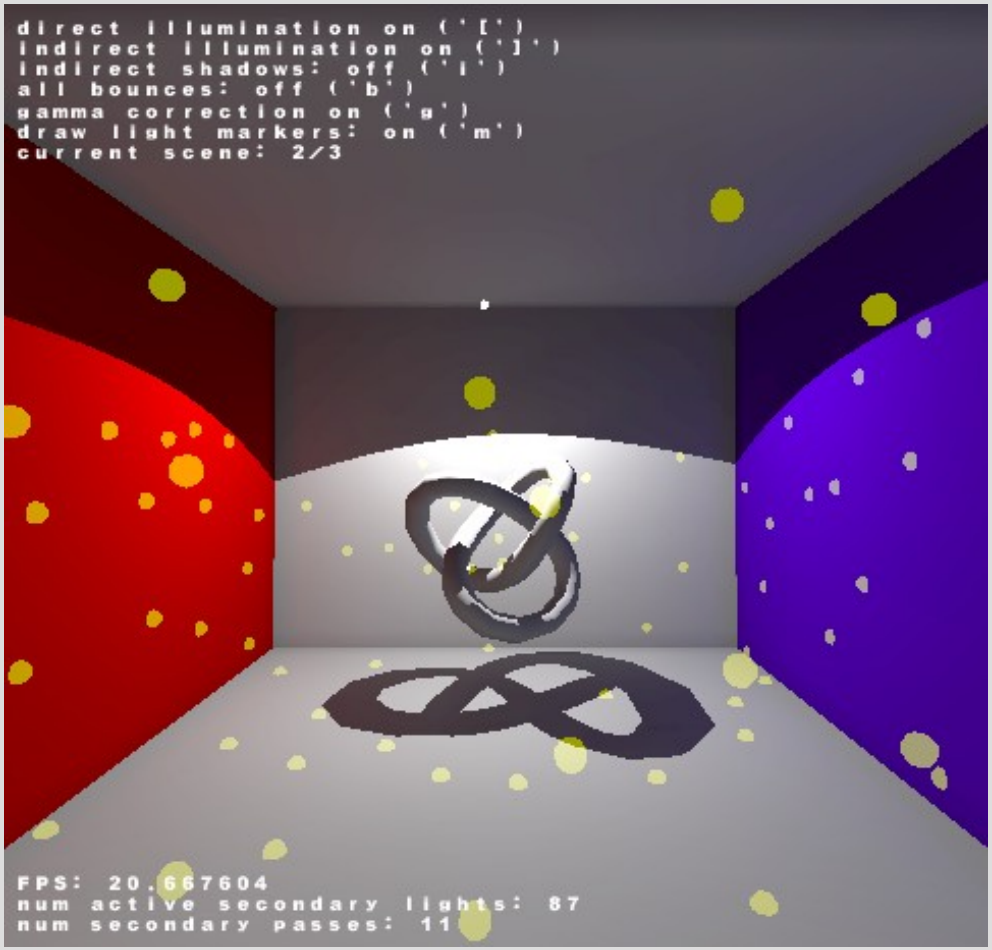
- ***Instant Radiosity***

Alexander Keller









Aceleración por GPU

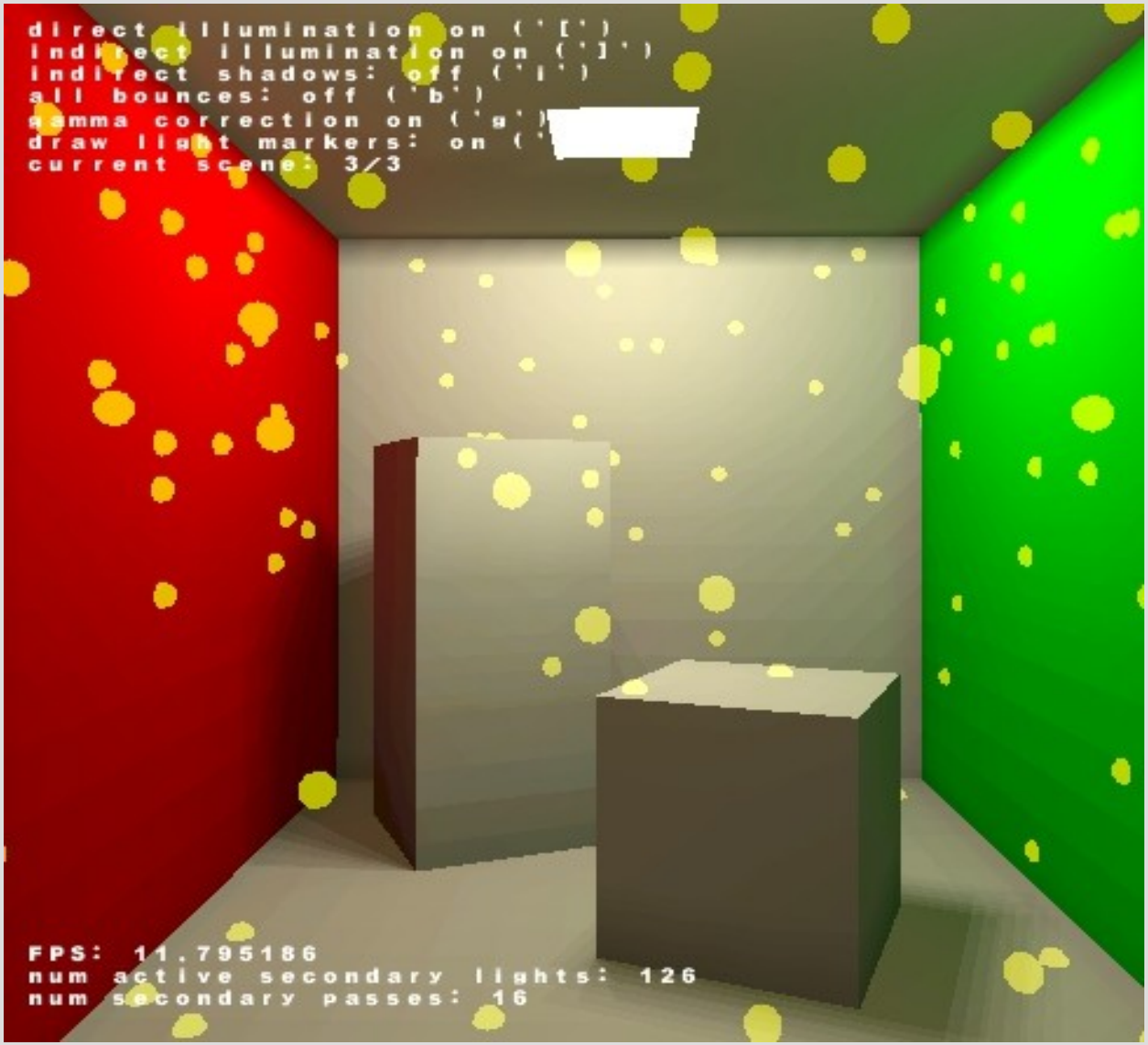
- Puntos de luz virtuales

Aceleración por GPU

- Algoritmo:

```
InstantRadiosity()
        GenerarPartículas();
        UbicarLucesPuntuales();
        IntensidadLucesPuntuales();
        DesplegarIluminaciónDirecta();
        DesplegarIluminaciónIndirecta();
FinInstantRadiosity();
```

Aceleración por GPU



Aceleración por GPU

- **Sistema de ecuaciones en el *Fragment Shader*.**

$$\begin{bmatrix} 1-\rho_1 F_{1,1} & -\rho_1 F_{1,2} & \cdots & -\rho_1 F_{1,N} \\ -\rho_2 F_{2,1} & 1-\rho_2 F_{2,2} & \cdots & -\rho_2 F_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_N F_{N,1} & -\rho_N F_{N,2} & \cdots & 1-\rho_N F_{N,N} \end{bmatrix} x \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_N \end{bmatrix}$$

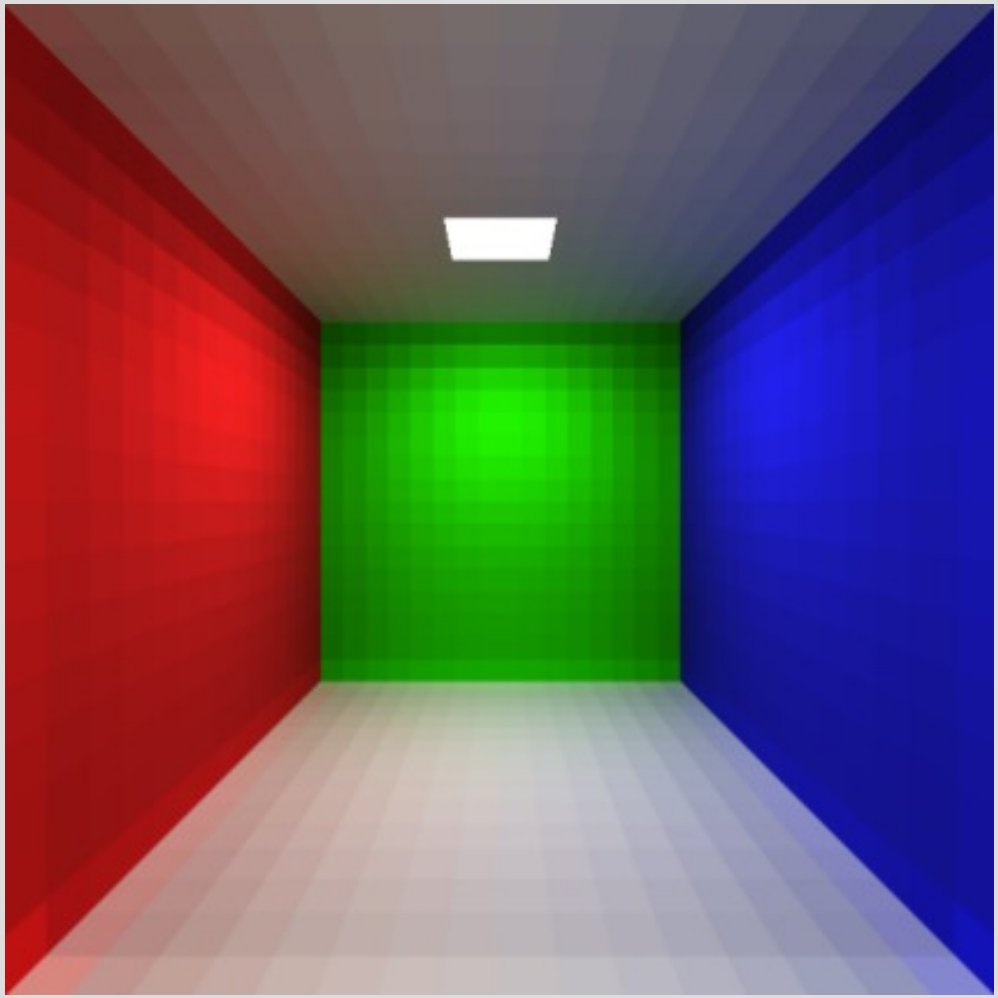
- *RGB*
- *Luminance*

$$B^{(k+1)} = B^{(k)} + E - diag(M)^{-1} M B^{(k)}$$

Aceleración por GPU

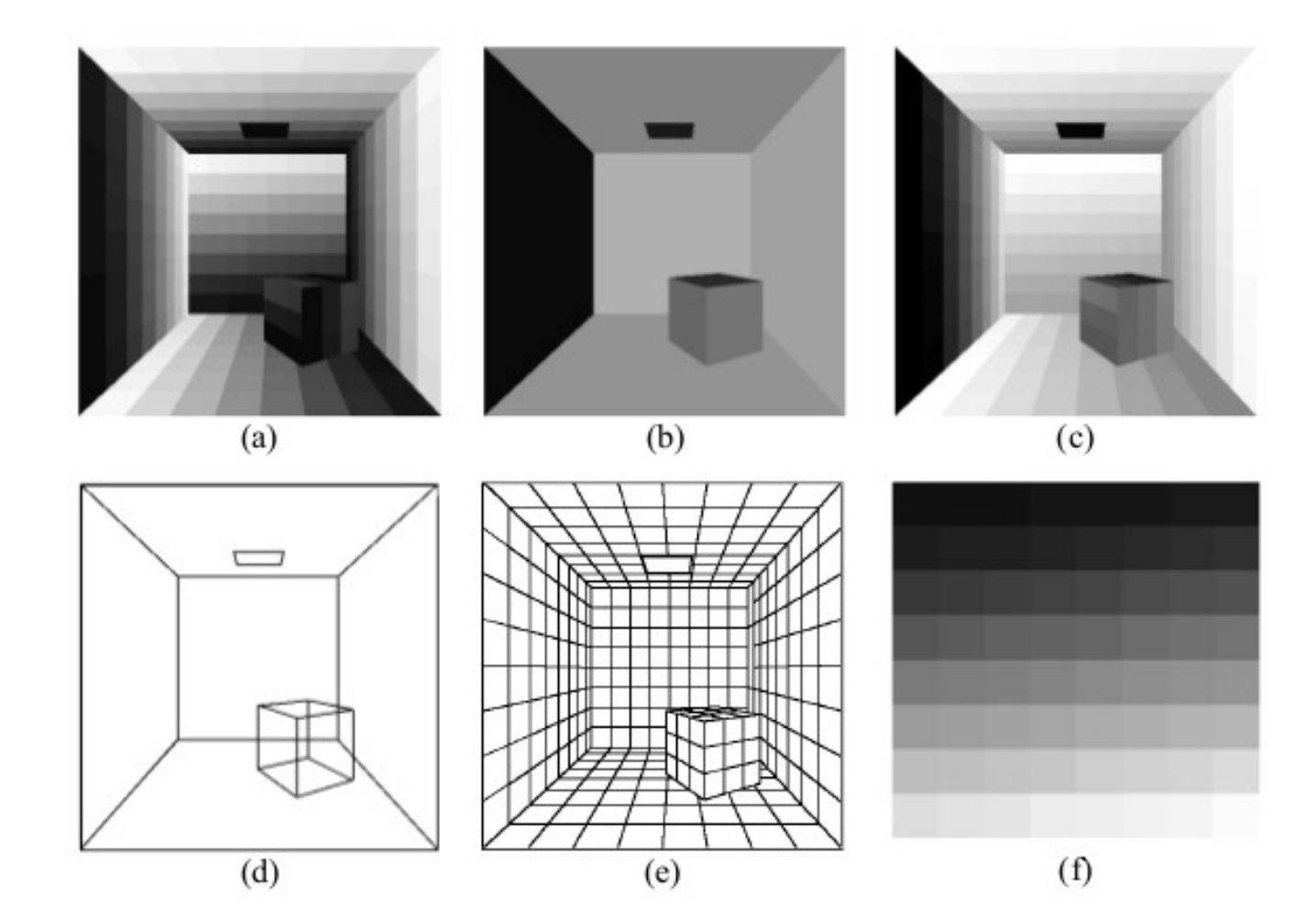
- Resultados:

Nathan A. Carr



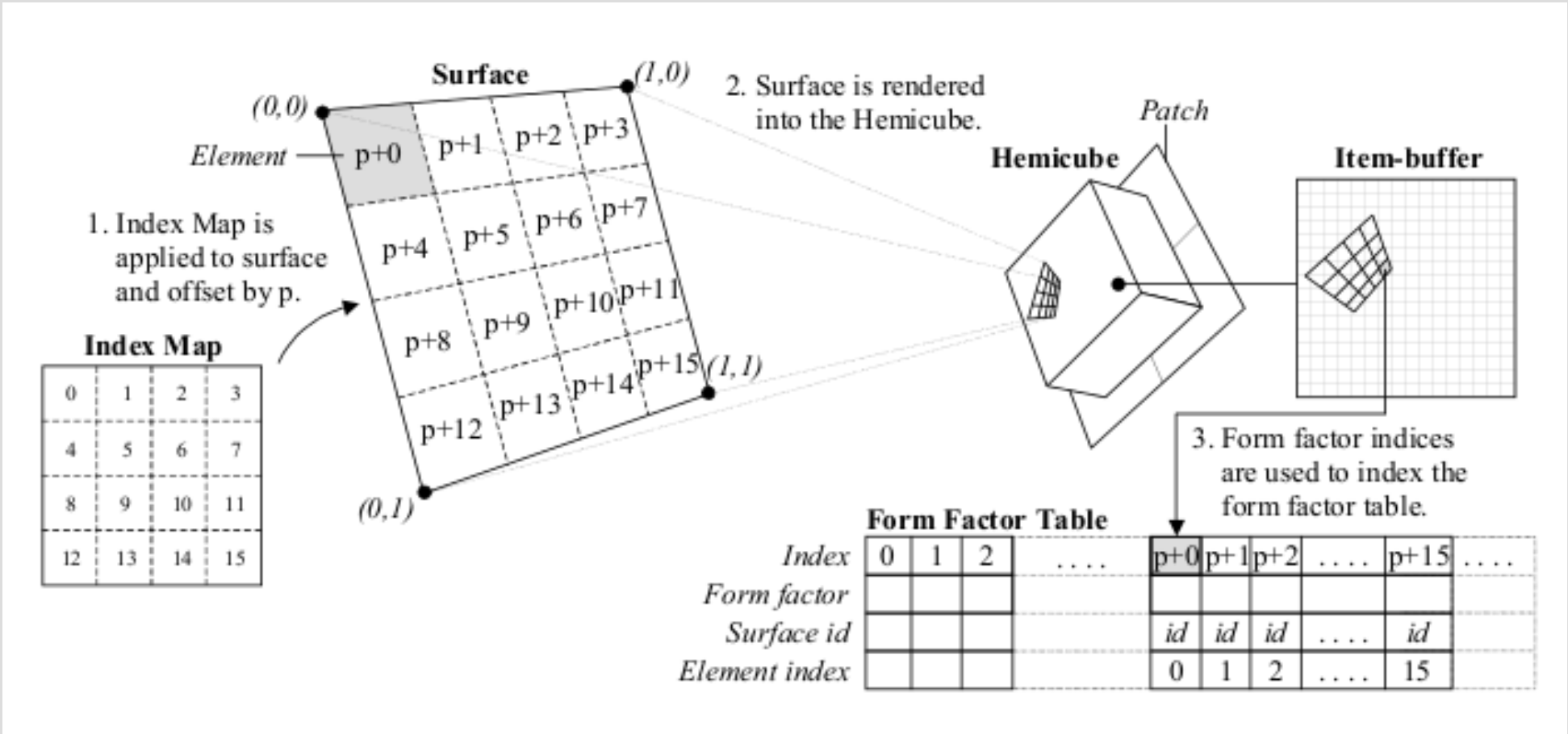
Aceleración por GPU

- **Mapas de índices**.



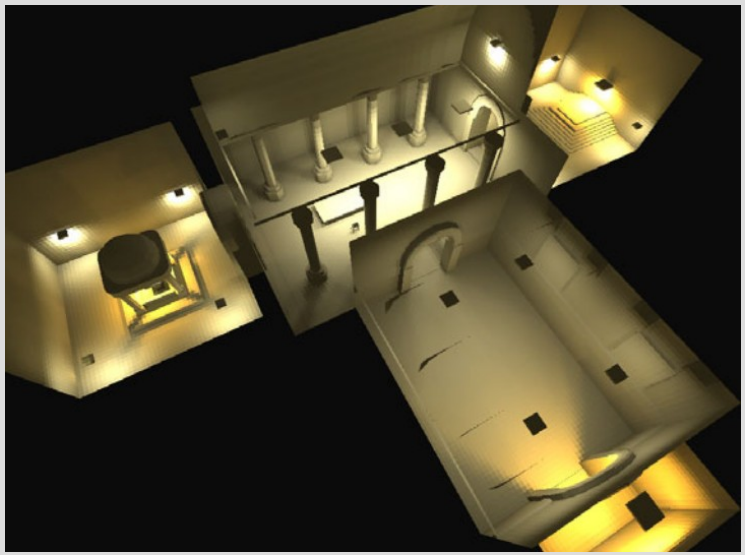
Aceleración por GPU

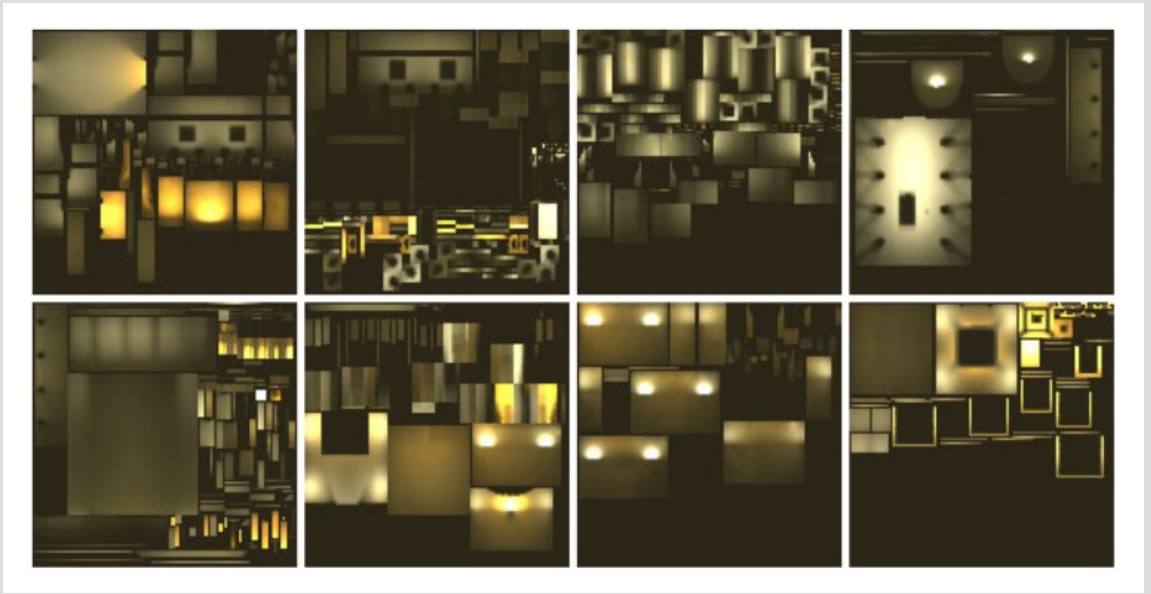
- Semicubos modificados

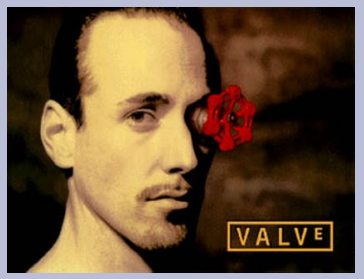


Aceleración por GPU

- Resultados:







Formas de simular *Radiosity*

- ***Radiosity Normal Mapping* (VALVE)**

Gordon Freeman

Normal-mapped

Normal mapped with ambient occlusion

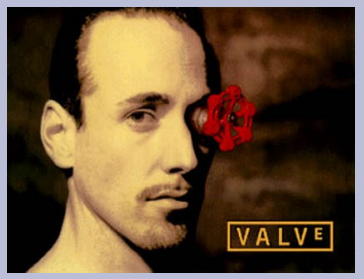
Self-shadowed

Height map

Bump map

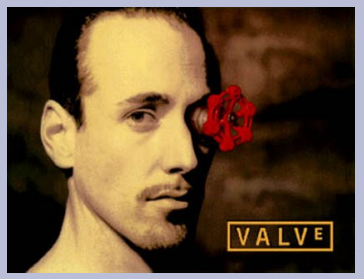
Bump map stored in new basis

With directional ambient occlusion

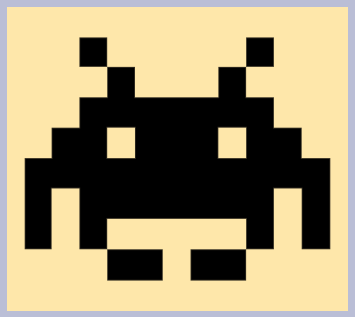


Formas de simular *Radiosity*

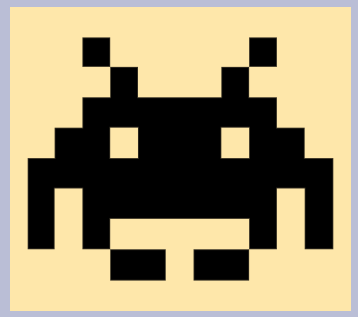
- Resultados:



Formas de simular *Radiosity*



Demostraciones



Demostraciones

Fast Radiosity on the GPU

- **Conclusiones**:
  - *Radiosity* es una técnica de iluminación global que produce muy buenos resultados y es de fácil implementación (relativamente).
  - La naturaleza del algoritmo de *Radiosity*, tanto su formulación clásica como el refinamiento progresivo pueden acelerarse aprovechando la arquitectura paralela del GPU.
  - *Radiosity* en tiempo real es posible.

¿Preguntas?