Podstawy Grafiki Komputerowej

Wykład 8: Ray Tracing + Global Illumination

Andrzej Łukaszewski

Pracownia Grafiki Komputerowej
(LIGHT — Laboratory of Imaging and GrapHic Techniques)

Instytut Informatyki, Uniwersytet Wrocławski

2 grudnia 2019

Ray Tracing: śledzenie promieni

- Nierekurencyjna metoda dla widoczności (ray casting) — Appel, 1968
- Rekurencyjny ray–tracing — Whitted, 1980

- Metody Monte Carlo — symulujące fizycznie oświetlenie

Ray Tracing: rekurenc. śledzenie promieni

```
1  dla każdego piksela                          // primary ray
2     oblicz promień R od obserwatora przez ekran
3     kolor piksela =  RayTrace(R)
4
5  Funkcja RayTrace(promien R) zwraca kolor
6     Znajdź przecięcie z najbliższym obiektem
7     // (czyli: t, punkt, id/material, wekt.normalny)
8     kolor = czarny (lub kolor tła)
9     jeśli jest przecięcie
0        dla każdego źródła światła       // shadow ray
1           wygeneruj promień od punktu do światła
2           czy promień przecina się z którymś obiektem
3           jeśli nie ma przecięcia       // in light
4             uaktualnij kolor o oświetlenia dla światła
5        wygeneruj promień odbity-zwierciadlany Refl
6        kolor_odbicia = RayTrace(Refl)  // rekurencja
7        uaktualnij kolor o kolor odbicia
```

Ray Tracing: śledzenie promieni

Algorytm bardzo prosty i łatwy do rozbudowy o kolejne efekty:

- refrakcja
- symulacja obiektywu — głębia ostrości
- rozmycie ruchu
- światła powierzchniowe ? inne odbicia ?

Koszt działania był uważany za duży ale wraz z rozwojem algorytmów i szybkości komputerów i rosnącymi scenami jest coraz atrakcyjniejszy ze względu na:

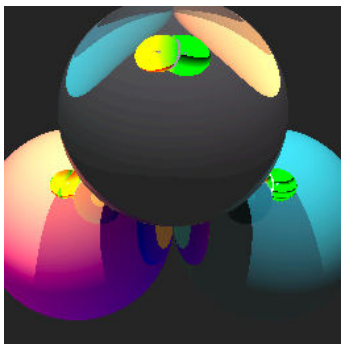
- efektywne metody znajdowania przecięć
- struktury przyspieszające dające koszt logarytmiczny względem ilości obiektów.

Ray Tracing: śledzenie promieni

Minimal Ray Tracing Competition:

jeden ze zwycięzców: 80x25 znaków (na wizytówkę)

```
typedef struct{double x,y,z}vec;vec U,black,amb={.02,.02,.02};struct sphere{
vec cen,color;double rad,kd,ks,kt,kl,ir}*s,*best,sph[]={0.,6.,.5,1.,1.,1.,.9,
.05,.2,.85,0.,1.7,-1.,8.,-.5,1.,.5,.2,1.,.7,.3,0.,.05,1.2,1.,8.,-.5,.1,.8,.8,
1.,.3,.7,0.,0.,1.2,3.,-6.,15.,1.,.8,1.,7.,0.,0.,0.,.6,1.5,-3.,-3.,12.,.8,1.,
1.,5.,0.,0.,0.,.5,1.5,};yx;double u,b,tmin,sqrt(),tan();double vdot(A,B)vec A
,B;{return A.x*B.x+A.y*B.y+A.z*B.z;}vec vcomb(a,A,B)double a;vec A,B;{B.x+=a*
A.x;B.y+=a*A.y;B.z+=a*A.z;return B;}vec vunit(A)vec A;{return vcomb(1./sqrt(
vdot(A,A)),A,black);}struct sphere*intersect(P,D)vec P,D;{best=0;tmin=1e30;s=
sph+5;while(s-->sph)b=vdot(D,U=vcomb(-1.,P,s->cen)),u=b*b-vdot(U,U)+s->rad*s
->rad,u=u>0?sqrt(u):1e31,u=b-u>1e-7?b-u:b+u,tmin=u>=1e-7&&u<tmin?best=s,u:
tmin;return best;}vec trace(level,P,D)vec P,D;{double d,eta,e;vec N,color;
struct sphere*s,*l;if(!level--)return black;if(s=intersect(P,D));else return
amb;color=amb;eta=s->ir;d= -vdot(D,N=vunit(vcomb(-1.,P=vcomb(tmin,D,P),s->cen
)));if(d<0)N=vcomb(-1.,N,black),eta=1/eta,d= -d;l=sph+5;while(l-->sph)if((e=l
->kl*vdot(N,U=vunit(vcomb(-1.,P,l->cen))))>0&&intersect(P,U)==l)color=vcomb(e
,l->color,color);U=s->color;color.x*=U.x;color.y*=U.y;color.z*=U.z;e=1-eta*
eta*(1-d*d);return vcomb(s->kt,e>0?trace(level,P,vcomb(eta,D,vcomb(eta*d-sqrt
(e),N,black))):black,vcomb(s->ks,trace(level,P,vcomb(2*d,N,D)),vcomb(s->kd,
color,vcomb(s->kl,U,black))));}main(){printf("P3\n%d %d\n255\n",32,32);while(yx<32*32)
U.x=yx%32-32/2,U.z=32/2-yx++/32,U.y=32/2/tan(25/114.5915590261),U=vcomb(255.,
trace(3,black,vunit(U)),black),printf("%.0f %.0f %.0f\n",U);}/*minray!*/
```



P.Heckbert - Graphics Gems IV, 1994

Ray Tracing: przecięcia z obiektami

Znajdowanie przecięcia promienia:

$$R(t) = O + t \cdot D$$

z obiektami:

- sfery i inne powierzchnie uwikłane $F(x, y, z) = 0$
- trójkąty
- prostopadłościany
- powierzchnie parametryczne
- inne reprezentacje

Ray Tracing: przecięcie z trójkątem

Algorytm Möller'a (1997): bez współ. równania płaszczyzny

$$O + tD = (1 - u - v)V_0 + uV_1 + vV_2$$

$$[-D, V_1 - V_0, V_2 - V_0] \begin{bmatrix} t \\ u \\ v \end{bmatrix} = O - V_0$$

Rozwiązanie z wzorów Crammer'a:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{|-D, E_1, E_2|} \begin{bmatrix} |T, & E_1, & E_2| \\ |-D, & T, & E_2) \\ |-D, & E_1, & T| \end{bmatrix}$$

Korzystając z: $|A, B, C| = -(A \times B) \cdot B = -(C \times B) \cdot A$. mamy:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{(D \times E_2) \cdot E_1} \begin{bmatrix} (T \times E_1) \cdot E_2 \\ (D \times E_2) \cdot T \\ (T \times E_1) \cdot D \end{bmatrix} = \frac{1}{P \cdot E_1} \begin{bmatrix} Q \cdot E_2 \\ P \cdot T \\ Q \cdot D \end{bmatrix}$$

Koszt: 1 dzielenie, 27 mnożeń, 17 dodawań/odejmowań

Ray Tracing: przecięcie z trójkątem

Oryginalny kod ze strony Tomasa Akenine-Möller'a

```
#define EPSILON 0.000001
#define CROSS(dest,v1,v2) \
          dest[0]=v1[1]*v2[2]-v1[2]*v2[1]; \
          dest[1]=v1[2]*v2[0]-v1[0]*v2[2]; \
          dest[2]=v1[0]*v2[1]-v1[1]*v2[0];
#define DOT(v1,v2) (v1[0]*v2[0]+v1[1]*v2[1]+v1[2]*v2[2]
#define SUB(dest,v1,v2)
          dest[0]=v1[0]-v2[0]; \
          dest[1]=v1[1]-v2[1]; \
          dest[2]=v1[2]-v2[2];
```

Ray Tracing: przecięcie z trójkątem

```
int intersect_triangle(double orig[3], double dir[3],
                       double vert0[3], double vert1[3], double vert2[3],
                       double *t, double *u, double *v) {
   double edge1[3], edge2[3], tvec[3], pvec[3], qvec[3];
   double det,inv_det;

   /* find vectors for two edges sharing vert0 */
   SUB(edge1, vert1, vert0);
   SUB(edge2, vert2, vert0);

   /* begin calculating determinant - also used to calculate U parameter */
   CROSS(pvec, dir, edge2);

   /* if determinant is near zero, ray lies in plane of triangle */
   det = DOT(edge1, pvec);

#ifdef TEST_CULL           /* define TEST_CULL if culling is desired */
   if (det < EPSILON)
      return 0;

   /* calculate distance from vert0 to ray origin */
   SUB(tvec, orig, vert0);

   /* calculate U parameter and test bounds */
   *u = DOT(tvec, pvec);
   if (*u < 0.0 || *u > det)           return 0;

   /* prepare to test V parameter */
   CROSS(qvec, tvec, edge1);

    /* calculate V parameter and test bounds */
   *v = DOT(dir, qvec);
   if (*v < 0.0 || *u + *v > det)      return 0;
```

Ray Tracing: przecięcie z trójkątem

```
   /* calculate t, scale parameters, ray intersects triangle */
   *t = DOT(edge2, qvec);
   inv_det = 1.0 / det;
   *t *= inv_det;
   *u *= inv_det;
   *v *= inv_det;
#else                    /* the non-culling branch */
   if (det > -EPSILON && det < EPSILON) return 0;
   inv_det = 1.0 / det;

   /* calculate distance from vert0 to ray origin */
   SUB(tvec, orig, vert0);

   /* calculate U parameter and test bounds */
   *u = DOT(tvec, pvec) * inv_det;
   if (*u < 0.0 || *u > 1.0)            return 0;

   /* prepare to test V parameter */
   CROSS(qvec, tvec, edge1);

   /* calculate V parameter and test bounds */
   *v = DOT(dir, qvec) * inv_det;
   if (*v < 0.0 || *u + *v > 1.0)       return 0;

   /* calculate t, ray intersects triangle */
   *t = DOT(edge2, qvec) * inv_det;
#endif
   return 1;
}
```

Ray Tracing: struktury przyspieszające

Aby nie liczyć przecięć z wszystkimi obiektami trawersujemy struktury sprawdzając tylko niektóre obiekty:

- kraty — grids (A.Fujimoto)
- kd–drzewa (M.Kaplan)
- BVH — bounding volume hierarchies

Różne heurystyki konstruowania optymalnych drzew:

mediana, SAH

Różne metody trawersowania: rekurencyjnie, iteracyjnie

Optymalizacje użycia pamięci:

np. węzły kd–drzewa na 8 bajtach, alignement

Ray Tracing: kd-drzewa (M.Kaplan 1985)

Drzewo dzieli przestrzeń niejednorodnie. Trawersowanie:

```
1  TreeIntersect(ray R, node W)
2    if  R.interval empty or W empty then return
3      if  W is leaf then
4         foreach object O from W
5            Intersect(R,O)
6      else
7         R1= R clipped to nearer side of W
8         TreeIntersect(R1, W.near)
9         if NoIntersectionInside then
0           R2= R oclipped to farther side of W
1           TreeIntersect(R2, W.far)
```

W praktyce średnia złożoność trawersowania to $O(logN)$, a konstrukcji $O(NlogN)$

Ray Tracing: BVH

W hierarchii brył otaczających każdy obiekt tylko raz !

Dzielimy obiekty, a nie przestrzeń.

```
1  BVH_Intersect(ray, node)
2    if node is a leaf
3      Intersect(ray, node.object)
4    else if Intersect_P(ray, node.BoundBox)
5      for each child of node do
6        BVH_Intersect(ray, child)
```

Nie można zakończyć trawersowania po pierwszym znalezionym przecięciu.

Które struktury nadają się dla animacji?

Ray Tracing: implementacje

Istnieje szereg dopracowanych bibliotek i kodu:

- Intel's Embree
- NVidia OptiX (CUDA)
- Kod z algorytmami z książki PBRT
- źródła innych rendererów dalej wymienianych np. Mitsuba

Jak otrzymać fotorealistyczny obraz ? fiz.?

Lokalny i globalny model oświetlenia

Globalny model oświetlenia: jednostki

Jednostki radiometryczne i fotometryczne:

Jednostka	Radiometria		Fotometria	Jednostka
Watt	Moc promienista	Φ	Strumień świetlny	lumen (lm)
$Watt/m^2$	Radiosity	B	Luminosity	Lux (lm/m^2)
$Watt/m^2$	Natężenie napromieniowania	E	Natężenie oświetlenia	Lux (lm/m^2)
$Watt/sr$	Radiant Intensity	I	Luminous Intensity	Candela(cd, lm/sr)
$Watt/m^2 sr$	Radiancja	L	Luminancja	Nit (cd/m^2, $lm/m^2 sr$)

Candela : 1/683 Watt/sr lub 18.3988 mW równo po całej sferze mocy wypromieniowanej o częstotliwości 540 THz (555.17 nm, żółto–zielony).

Radiancja/luminancja to wielkość propagowana po promieniach, postrzegana, odczytywana przez CCD,...

Funkcje BRDF opisują odbicie światła

Oznaczenia:

ω_i — kierunek padania światła

ω_s — kierunek odbicia zwierciadlanego (od ω_i)

ω_r — kierunek nas interesujący (odbity)

$$f_r(\omega_i \rightarrow, \omega_r) = \frac{(d)L_r(\omega_r)}{L_i(\omega_i)cos\theta_i d\omega_i}$$

$$f_r(x, \theta_i, \phi_i, \theta_r, \phi_r)$$

Zasada Helmholza: Foton poruszający się po ścieżce poruszałby się w przeciwnym kierunku także po tej samej ścieżce:

$$f_r(\omega_r, \omega_i) = f_r(\omega_i, \omega_r)$$

Postulat zachowania energii:

$$\int_\Omega f_r(\omega_i, \omega_r) \cos\theta_i d\omega_i \leq 1$$

funkcja BRDF dla odbicia rozproszonego ???

Równania renderingu, Kajiya 1985

Aby wygenerować obraz należy obliczyć radiancje $L(x, \omega)$ z każdego widocznego punktu sceny w kierunku kamery:

$$L(x, \omega_r) = \int_\Omega f_r(\omega_i, \omega_r) L_i(x, \omega_i) \cos\theta_i \, d\omega_i$$

Rekurencyjnie musimy policzyć radiancje dla większości punktów sceny i dowolnych kierunków.

Jeśli zdefiniujemy funkcje wzajemnej widoczności $V(x, y) = 0/1$ to ponieważ:

$$d\omega = dA \cdot \cos\theta / r^2$$

możemy zamienić całkowanie po kącie bryłowym na całkowanie po powierzchni sceny:

$$L_r(x, \omega_r) = \int_S f_r(\omega_i, \omega_r) L_r(y, \omega_y) V(x, y) \cos\theta_i \frac{\cos\theta_y \, dA}{r_{xy}^2}$$

Metody Monte Carlo

Nazwa Monte Carlo po raz pierwszy użyta dla metod matematycznych używanych w Los Alamos przy konstrukcji bomby w latach 40–tych (Neumann, Ulam, Fermi, Metropolis). Skuteczne szczególnie w przypadkach wielowymiarowych. Dla przestrzeni R_d o mierze 1:

— zwykłe kwadratury:

$$\int_{R^d} f(x_1, \ldots x_d)dx \approx \frac{1}{N^d} \sum_{i_1=1}^{N} \ldots \sum_{i_d=1}^{N} f(x_{i_1}^1, \ldots f_{i_d}^d)$$

— Monte Carlo:

$$\int_{R^d} f(x_1, \ldots x_d)dx \approx \frac{1}{N} \sum f(\xi_i^1, \ldots \xi_i^d)$$

Lit: M.H.Kalos, P.A.Whitlock – “Monte Carlo Methods, Volume I: Basics” John Wiley and Sons 1986

Metody Monte Carlo

Śledzenie ścieżek, mapy fotonów, dwukierunkowe śledzenie ścieżek,...
Efekty: oświetlenie pośrednie, krwawienie kolorów, kaustyki

Cornell Box

scena E. Veach'a

renderer: Mitsuba Jakoba Wenzel'a

Przykłady renderererów G.I.

W wielu programach do modelowania są dostępne renderery obliczające obraz fizycznie poprawnie zgodnie z globalnym modelem oświetlenia

Renderery opensourcowego Blender'a :

- Cycles (wbudowany),
- Pov-ray
- Yafray,
- aqsis
- Mitsuba
- LuxRender

Komercyjne programy: 3DStudio, Maya mają także wiele pluginów do wyboru: Mental Ray, Maxwell Renderer, ...

Można także zgadywać co jest fotografią, a co renderingiem: area.autodesk.com/fakeorfoto/, czasem jesteśmy bardzo blisko ideału.