Sphere tracing: integracja z klasycznymi metodami symulacji i renderingu

IGK 2012

Michał Jarząbek

W skrócie

- Funkcje niejawne – opisują powierzchnie niejawne
- Powierzchnie niejawne – metoda reprezentacji "obiektów" trójwymiarowych
- Funkcje odległości – pozbiór funkcji niejawnych
- Sphere tracing – metoda bazująca na śledzeniu promieni, służy do wizualizacji powierzchni reprezentowanych przez funkcje odległości

Powierzchnie niejawne

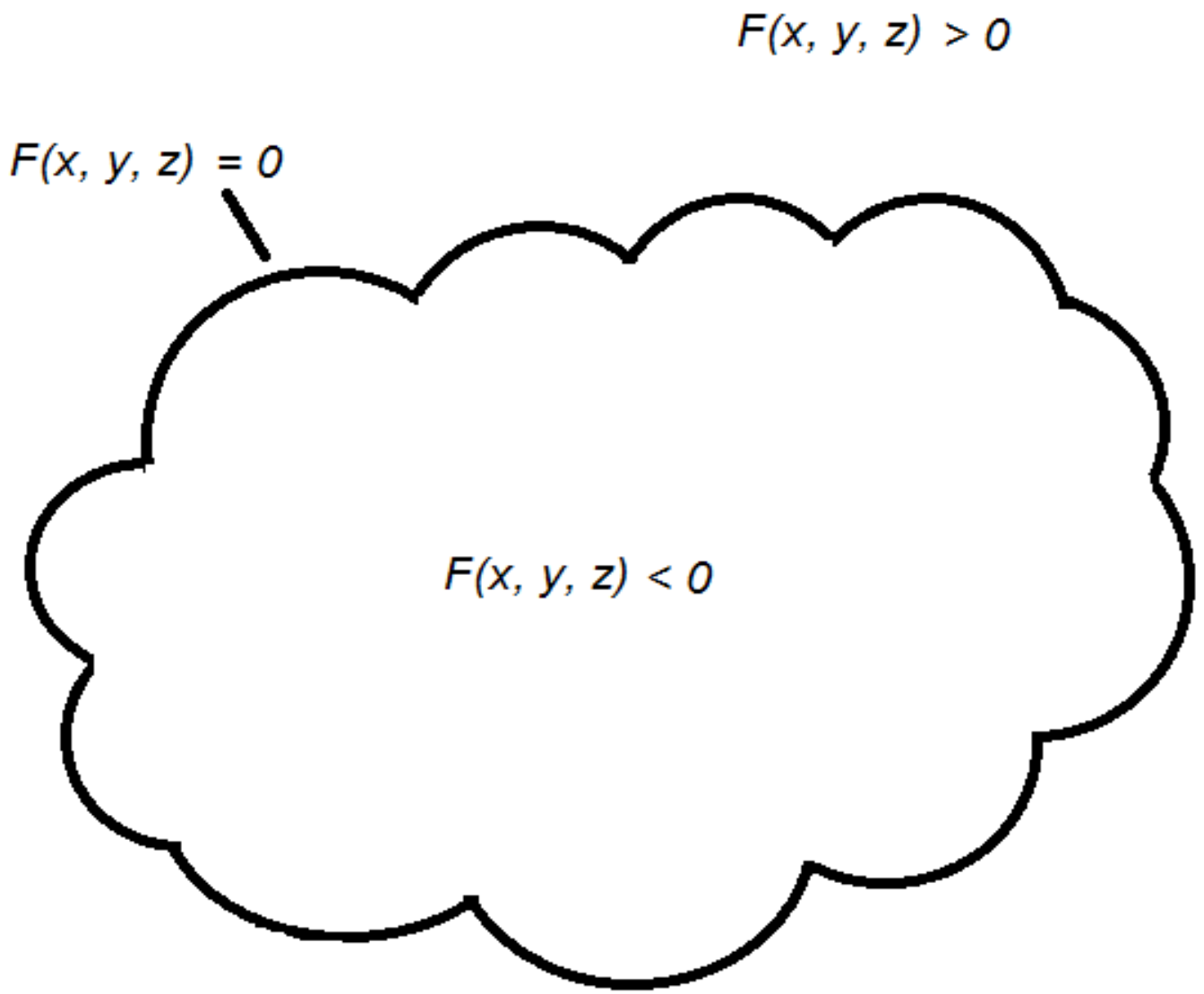
- Reprezentowane przez funkcje postaci

  $$F(x, y, z) = 0$$

  F to funkcja ciągła $R^3 \rightarrow R$.
- Przykład: równanie kuli

  $$x^2 + y^2 + z^2 - r^2 = 0$$

  dla każdego punktu poza powierzchnią kuli równanie nie jest prawdziwe
- Wartość funkcji określa czy punkt znajduje się wewnątrz, na, czy na zewnątrz powierzchni


F(x, y, z) > 0
F(x, y, z) = 0
F(x, y, z) < 0

Powierzchnie niejawne

- Reprezentowane przez funkcje postaci

  $$F(x, y, z) = 0$$

  F to funkcja ciągła $R^3 \rightarrow R$.
- Przykład: równanie kuli

  $$x^2 + y^2 + z^2 - r^2 = 0$$

- Nie ma nic szczególnego w wartości 0 po prawej stronie równania

Powierzchnie niejawne

- Reprezentowane przez funkcje postaci

  $$F(x, y, z) = 0$$

  F to funkcja ciągła $R^3 \rightarrow R$.
- Przykład: równanie kuli (nie o promieniu r)

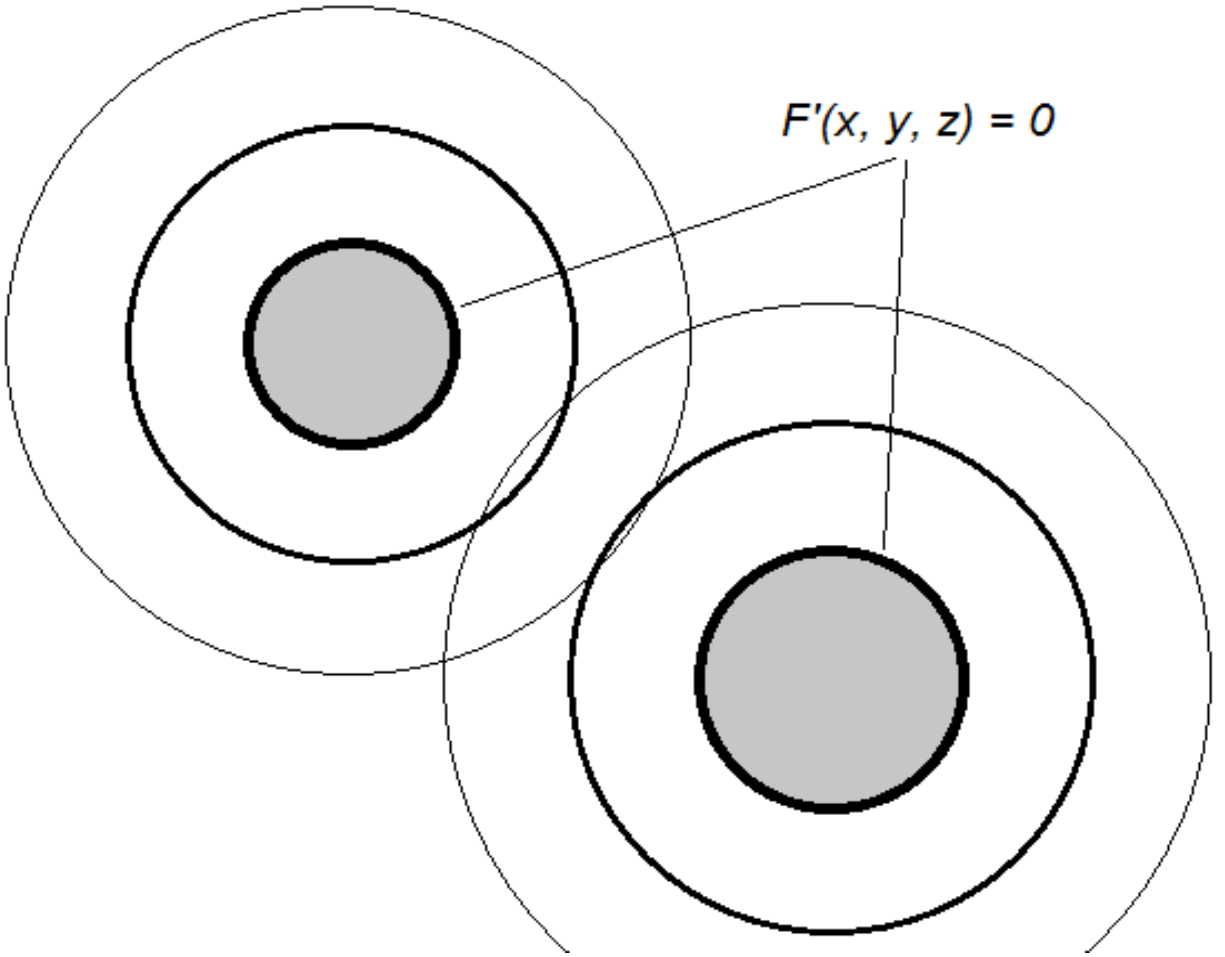
  $$x^2 + y^2 + z^2 - r^2 = T$$

  Zmieniając T otrzymamy inne powierzchnie – tzw. *izopowierzchnie* (na pierwszy rzut oka będą one większe/mniejsze, ale nie tylko)
- Możemy zgeneralizować:

  $$x^2 + y^2 + z^2 – r^2 – T = 0$$

  $$F(x, y, z) – T = 0$$

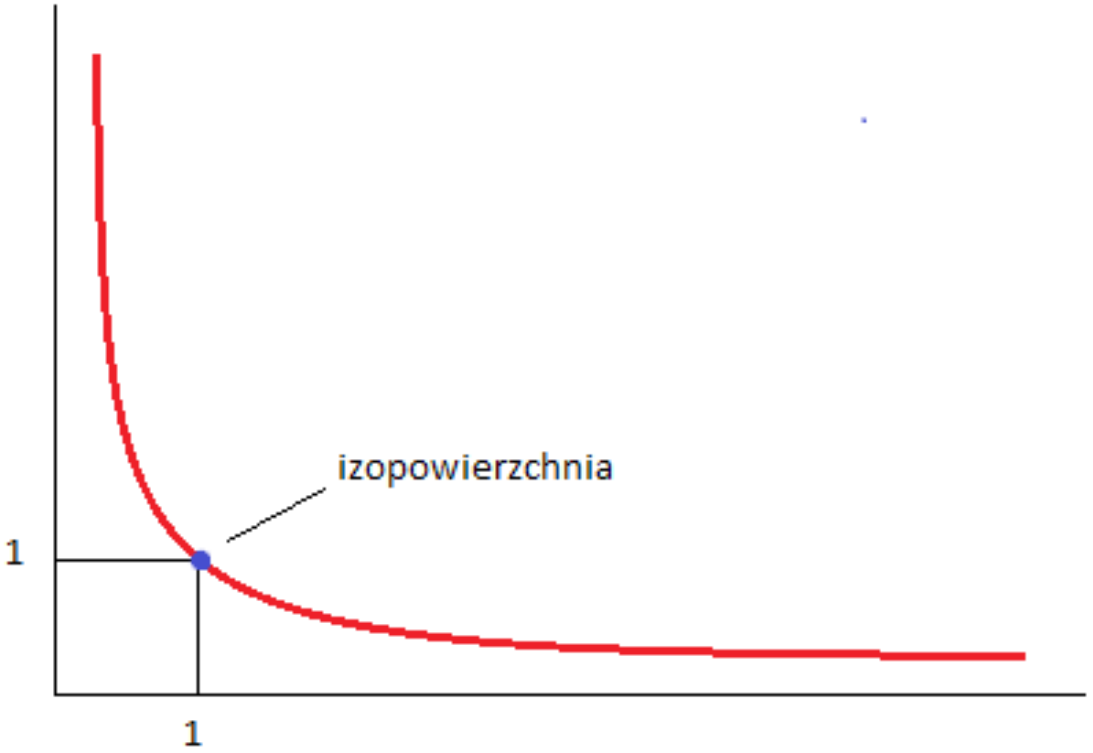
Fajniejsze powierzchnie niejawne



Pomyśl o środkach kuli jak o generatorach pola

Korekta!

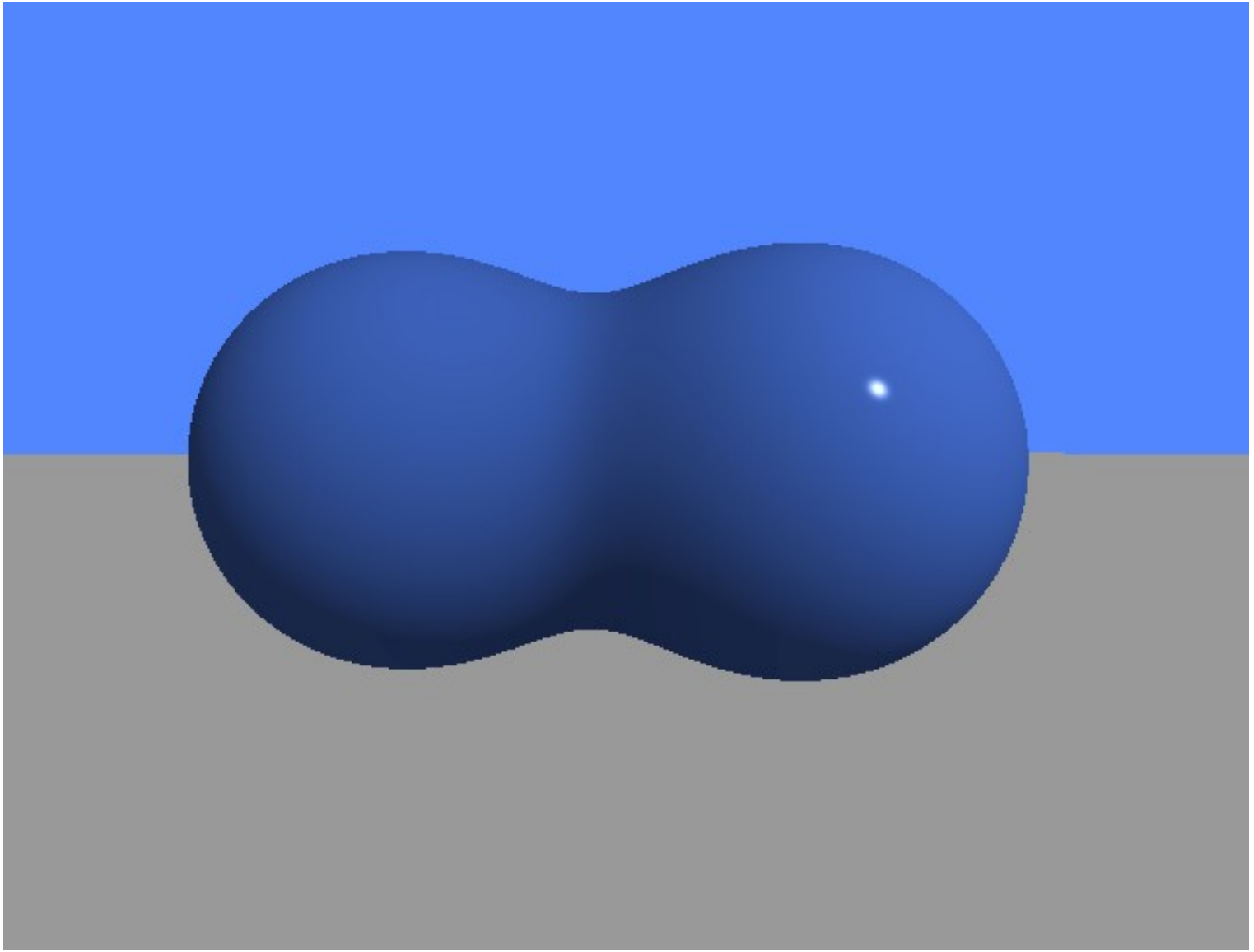
- Poprzedni sjald obrazuje zachowanie funkcji postaci:
  $f'(x, y, z) = 1 / ((x - sx)^2 + (y - sy)^2 + (z - sz)^2)$



Korekta!

- Poprzedni sjald obrazuje zachowanie funkcji postaci:
  $f'(x, y, z) = 1 / ((x - sx)^2 + (y - sy)^2 + (z – sz)^2)$
- Aby otrzymać takie zachowanie, należy:
  1. $f' + g'$
  2. $1 / (f' + g')$
  3. $1 / (f' + g') + T$ – to prawidłowa funkcja niejawna

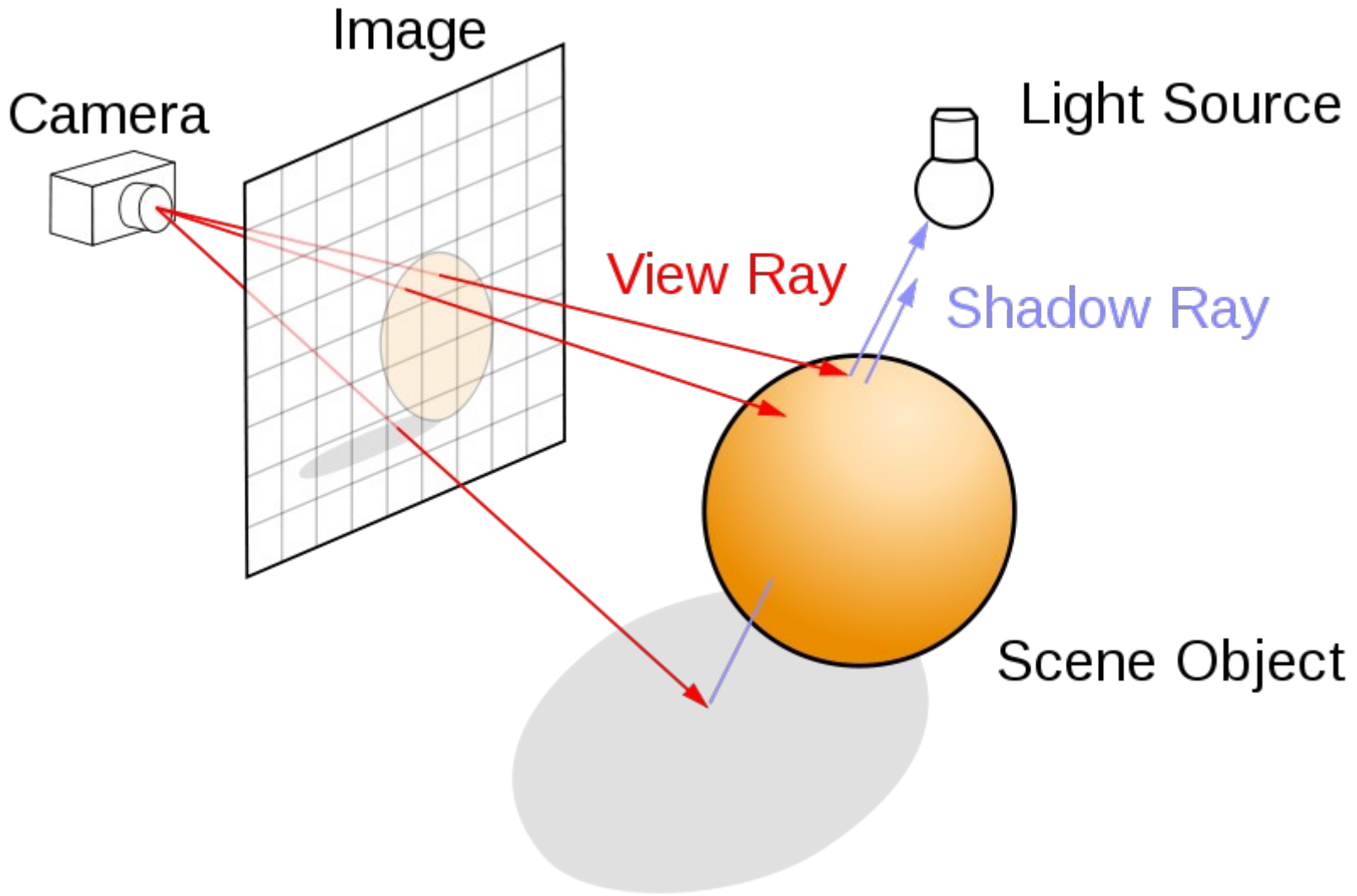
Po zbliżeniu ich do siebie: blob!



Jak to wyrenderować?!

- Wygenerować siatkę –
  np. algorytm Marching Cubes
- Ray tracing

Ray tracing

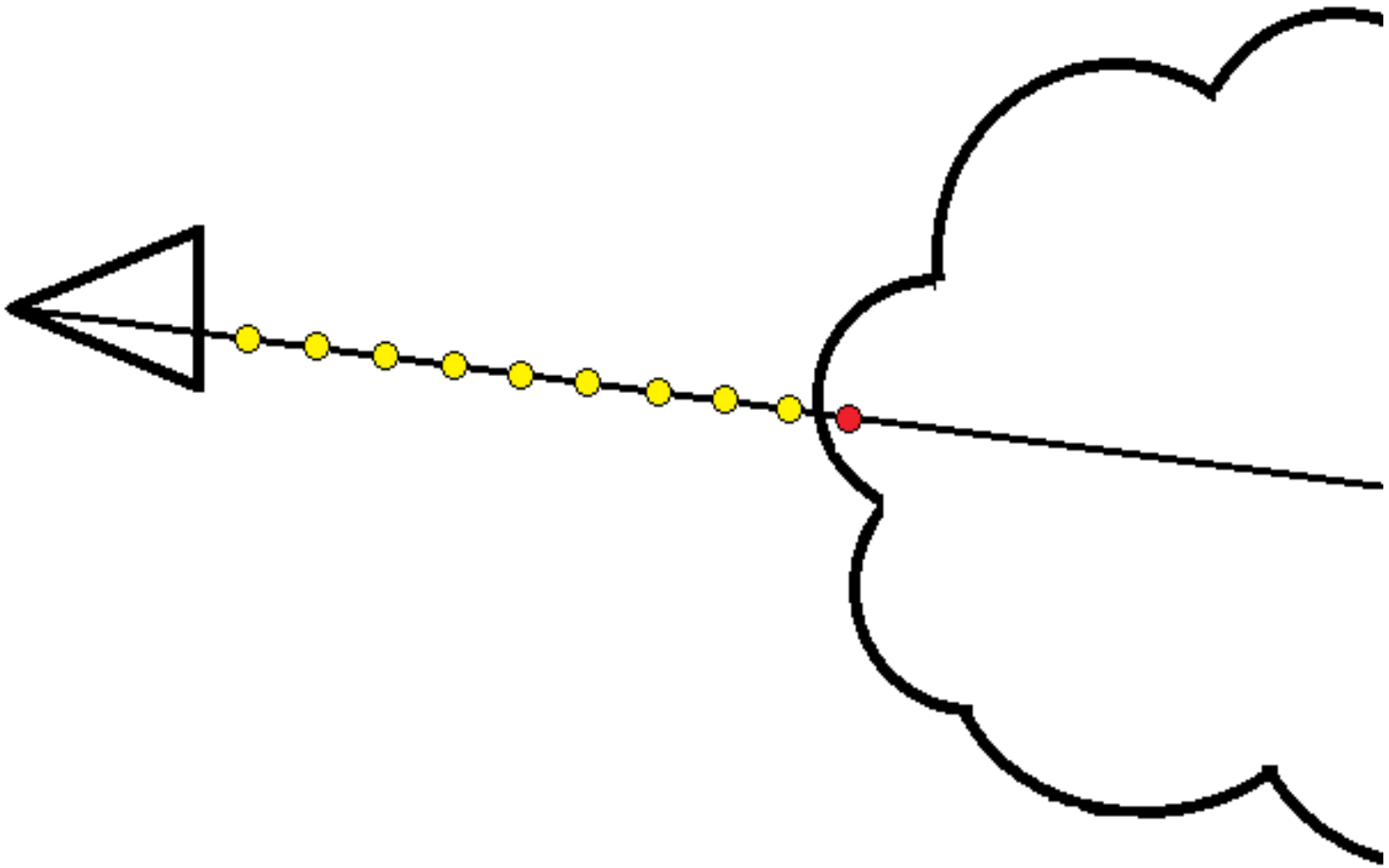


- Ukradziono z: wikipedia

Ray tracing

- Reprezentacja powierzchni musi umożliwiać znalezienie punktu przecięcia promienia powierzchnią
- Podstawić równanie promienia do równania powierzchni niejawnej
- Daje się rozwiązać tylko w przypadku prostych powierzchni

Ray marching



Ray marching

- Potrzebna możliwość sprawdzenia czy punkt na promieniu znajduje się na zewnątrz czy wewnątrz powierzchni – OK!

Ray marching

- Potrzebny wektor normalny
- Gradient funkcji (wektor pochodnych cząstkowych) jest prostopadły do powierzchni

$$\nabla F = \left(\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z} \right)$$

- Metoda różnic skończonych
- grad(func, p) = normalize (

  func(p+{eps,0,0}) - func(p-{eps,0,0}),

  func(p+{0,eps,0}) - func(p-{0,eps,0}),

  func(p+{0,0,eps}) - func(p-{0,0,eps}));

Ray marching

- Mamy punkt przecięcia i wektor normalny
- Oh so slow

Funkcje odległości

- Zwraca odległość do najbliższego punktu na powierzchni niejawnej
- Funkcje odległości ze znakiem (*signed distance function*) – podobnie, tylko dla punktów wewnatrz powierzchni zwraca wartość ujemną
- F.O. ze znakiem są podzbiorem powierzchni niejawnych
- Jednostajny wzrost

$$|\nabla d| = 1$$

Przykład

$F = x^2 + y^2 + z^2 - r^2$

- Nie jest prawidłową funkcją odległości
-

$d(\mathbf{x}) = ||\mathbf{x}|| - r$,

Gdzie:

$\mathbf{x} = (x, y, z)$,

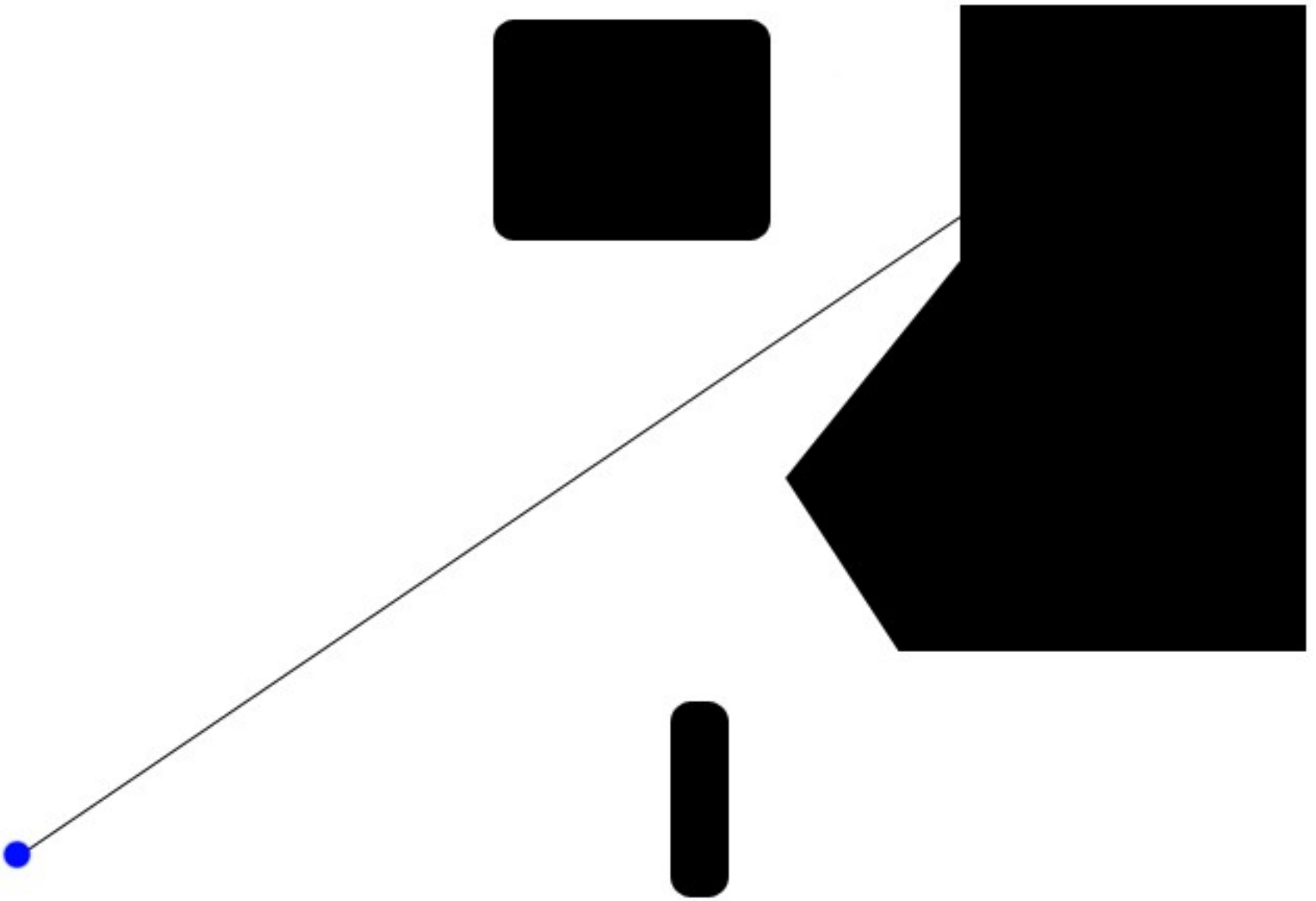
$||(x, y, z)||$ - norma euklidesowa

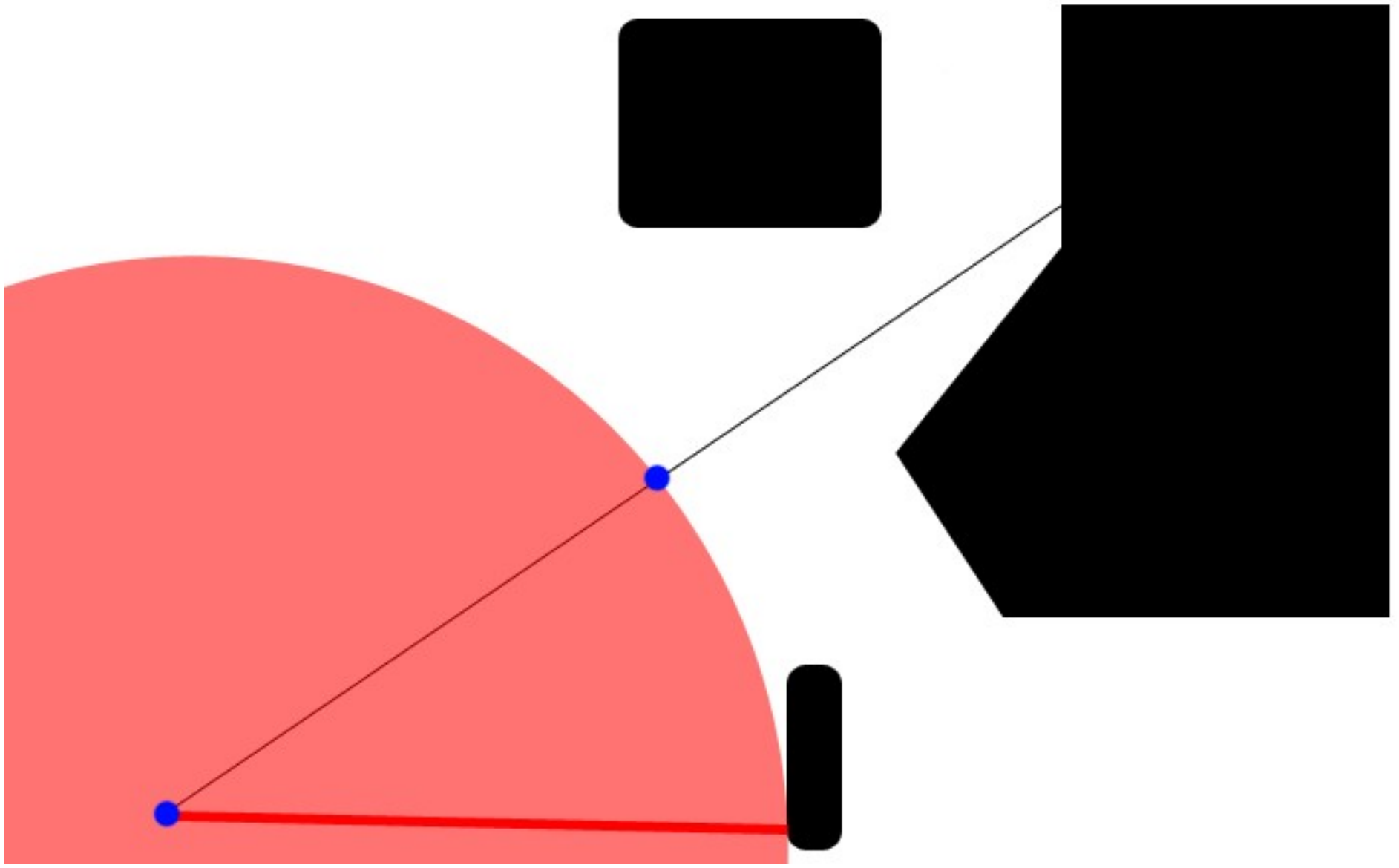
To prawidłowa funkcja odległości dla kuli

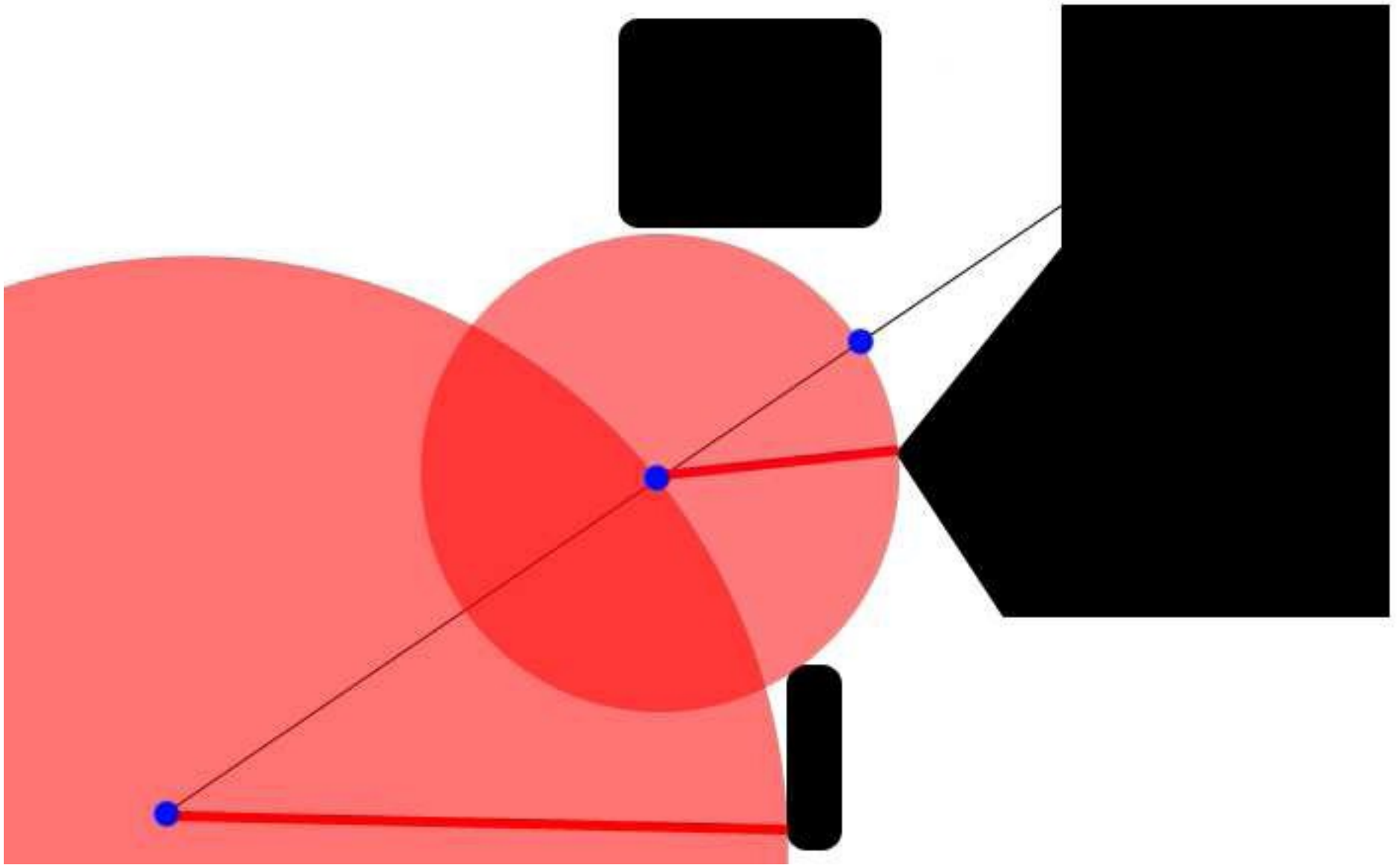
Sphere tracing

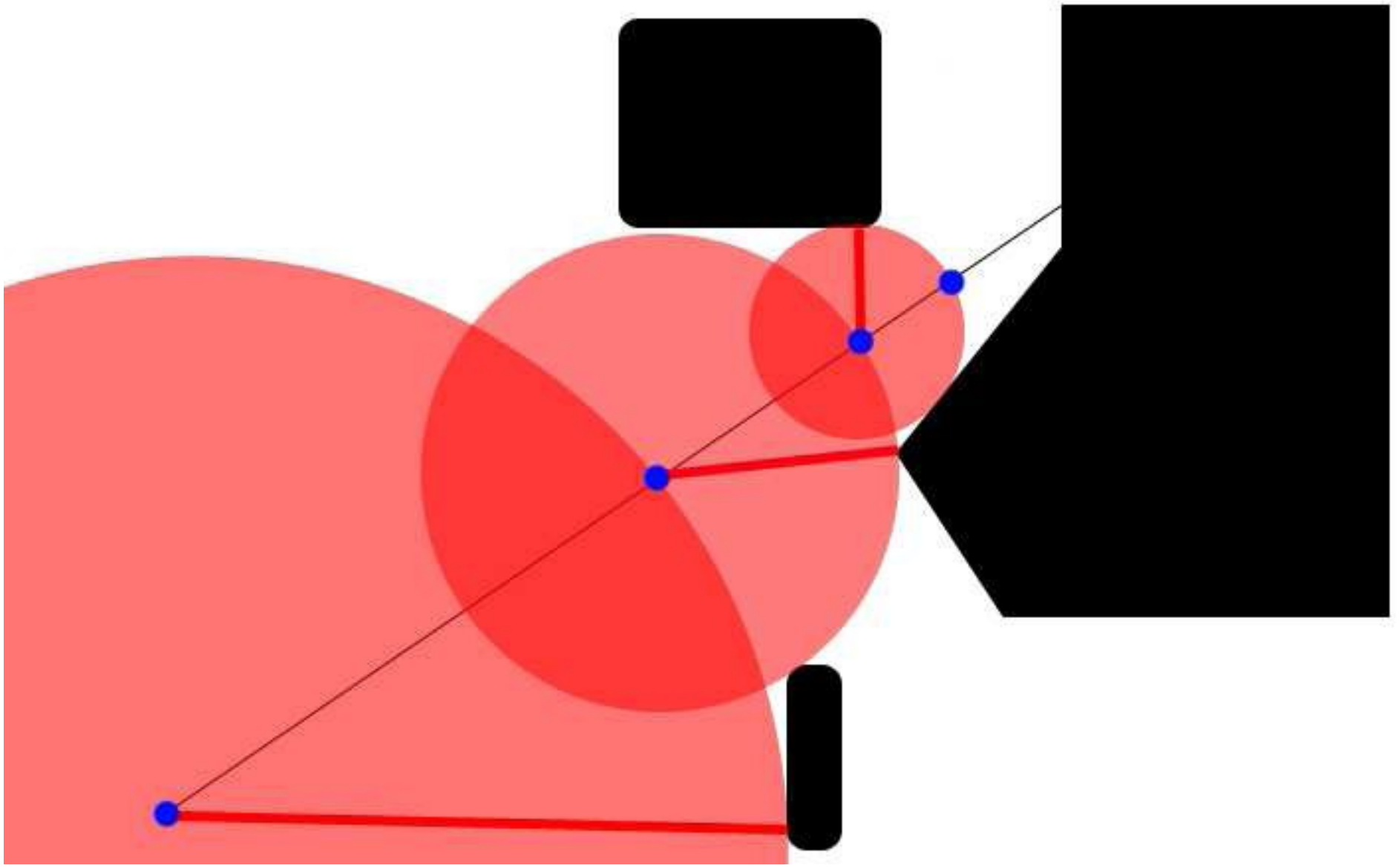
- Dzięki informacji o odległości do najbliższej powierzchni w danym punkcie możemy wykonywać znacznie większe kroki!

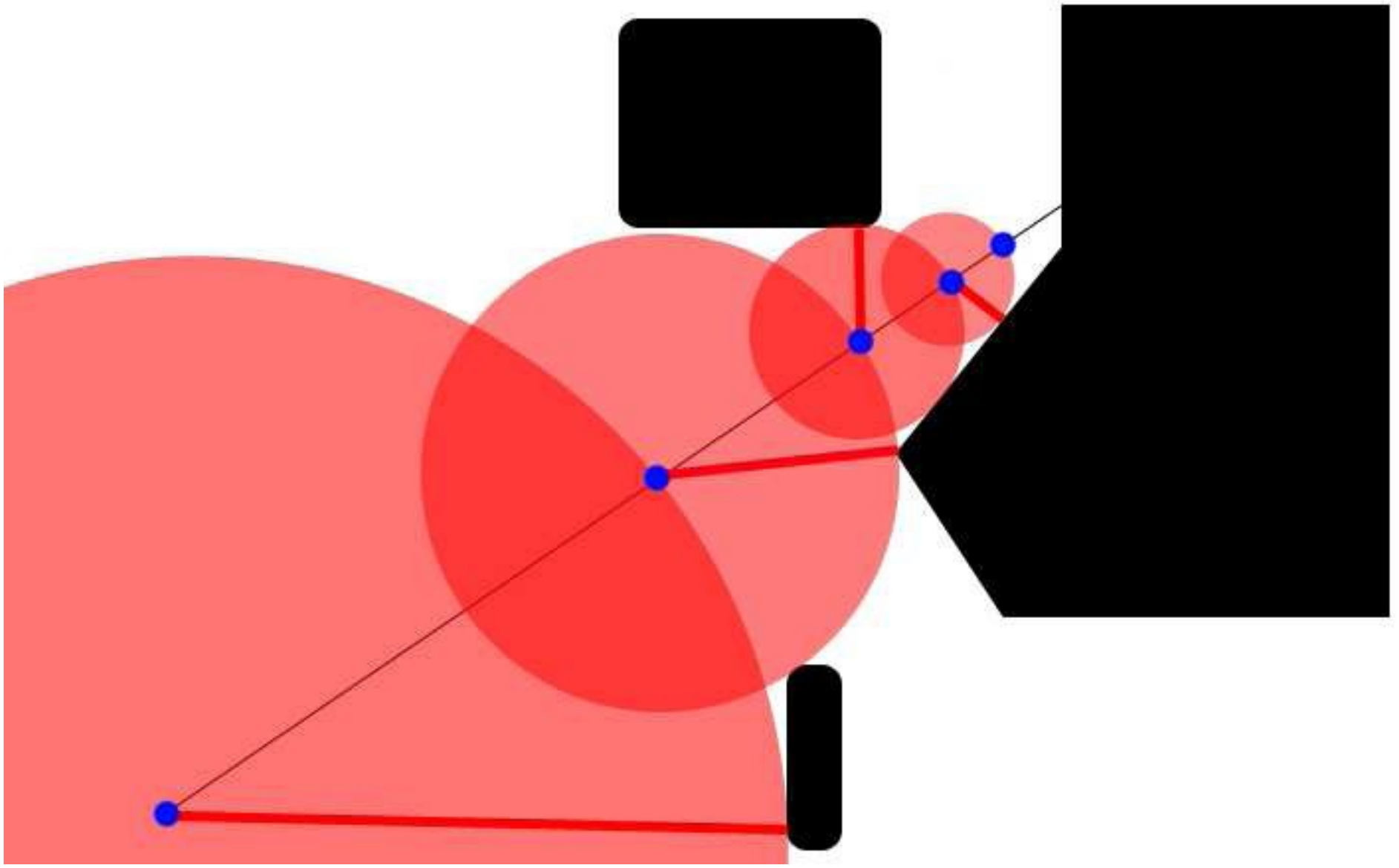
- Kolejne 7 slajdów dzięki uprzejmości Inigo Quilez http://www.iquilezles.org

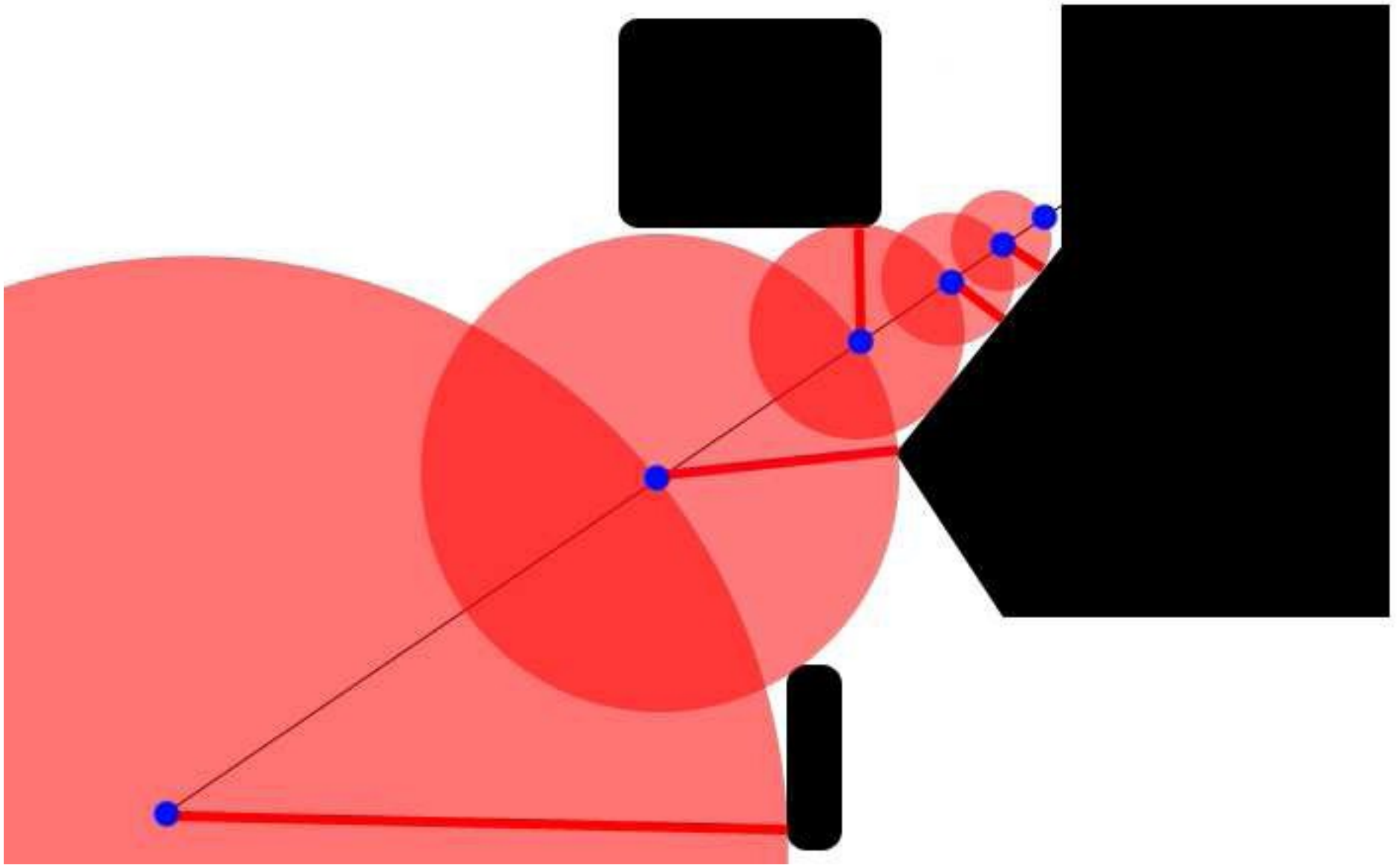


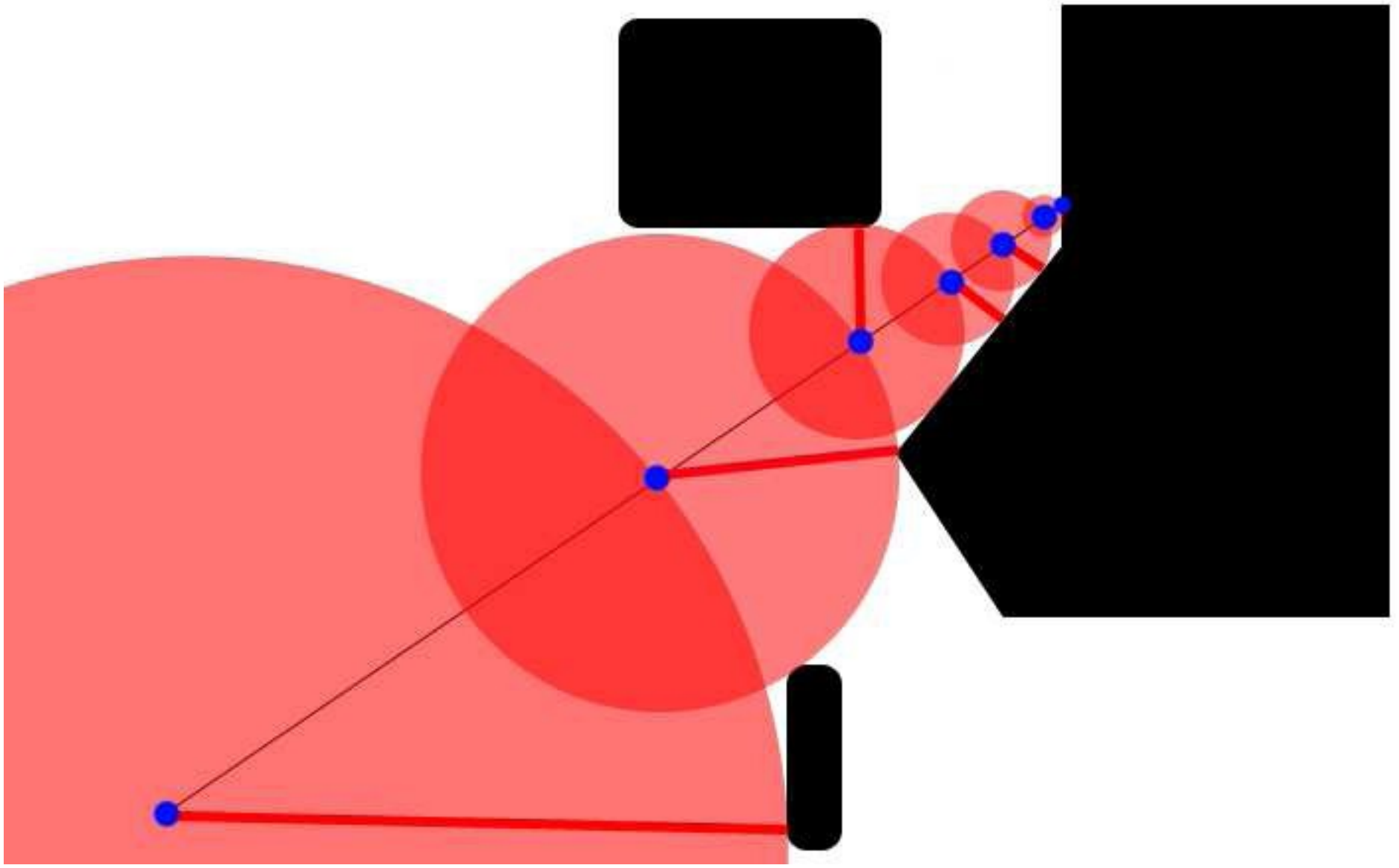












Sphere tracing

- Nie zawsze łatwo uzyskać idealną funkcję odległości
- Ograniczenie dolne na funkcję odległości też jest dobre!
- Trochę wolniej, ale poprawnie

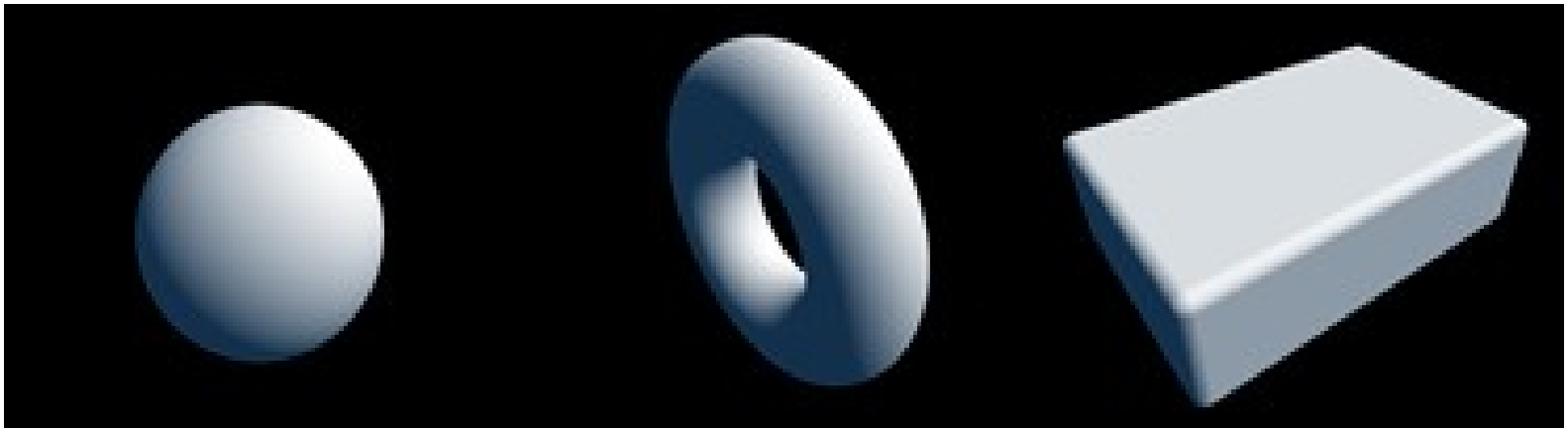
Modelowanie przy użyciu funkcji odległości

- Prymitywy geometryczne
- Funkcje odległości z ogólnych funkcji niejawnych
- CSG
- Transformacje
- Deformacje

Wzory na większość z poniższych znajdują się na stronie IQ:
http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm

Prymitywy geometryczne

- Proste procedury obliczające odległość powierzchni prymitywu ustawionego w środku układu współrzędnych od danego punktu

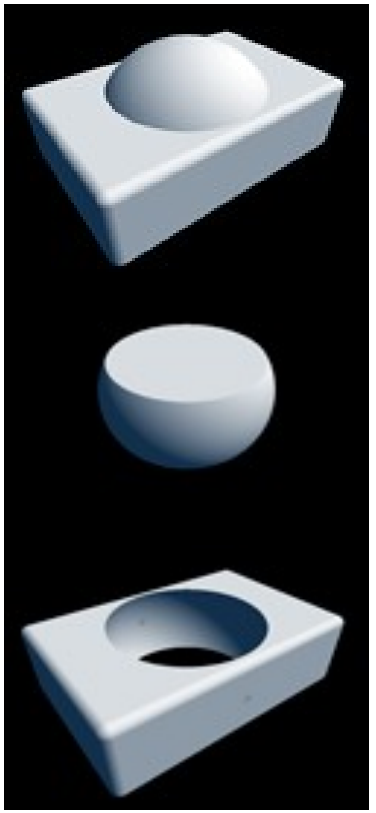


Funkcje odległości z ogólnych funkcji niejawnych

- Ciągłość Lipschitza ze stałą λ (globalnie lub na pewnym przedziale): $| f(\mathbf{x}) - f(\mathbf{y}) | <= \lambda | \mathbf{x} - \mathbf{y} |$
- Mówi o największej szybkości wzrostu funkcji na tym przedziale
- Porównaj: funkcje odległości mają dokładnie zdefiniowaną szybkość wzrostu funkcji (wielkość gradientu równa jeden)
- $d = f / \lambda$ – skalujemy f przez jej maks. wzrost
- d to ograniczenie dolne funkcji odległości powierzchni niejawnej opisanej przez f

Constructive solid geometry (CSG)

- Łączenie kilku powierzchni w jedną przy użyciu operacji boolowskich
- Mamy powierzchnie A i B i ich funkcje fA – fB
- Odległość od sumy A i B
  = min(fA(x, y, z), fB(x, y, z))
- Odległość od części wspólnej A i B
  = max(fA(x, y, z), fB(x, y, z))
- Odległość od różnicy A i B
  = max(fA(x, y, z), -fB(x, y, z))



Transformacje

- Jak dotąd – wszystko w środku układu współrzędnych
- Transformacja T = zastosowanie przekształcenia odwrotnego do dziedziny funkcji:

  $FT(x, y, z) = F(T^{-1}(x, y, z))$
- Translacja, rotacja – izometrie (zachowują odległość między punktami)
- Nie wszystkie interesujące przekształcenia to izometrie

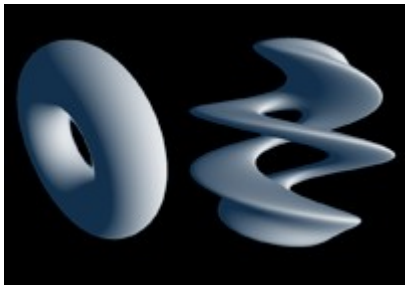
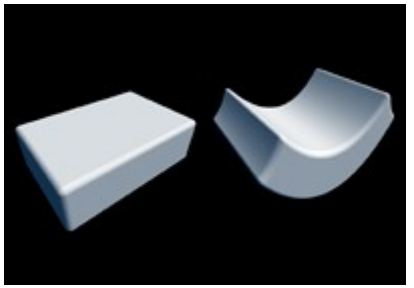
Transformacje

- Przekształcenia nieizometryczne – powodują, że funkcja przestaje być funkcją odległości
- Już widzieliśmy ten problem
- Należy znaleźć stałą Lipschitza odwrotności przekształcenia i podzielić przez nią funkcję FT
- Przykład: skalowanie jednorodne S ze współ. *s*
- Stała Lipschitza odwrotności = *1/s*

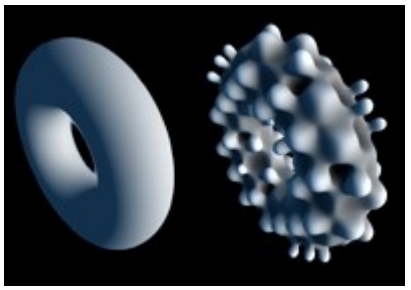
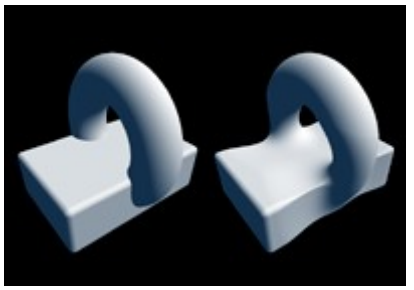
  $FS(x, y, z) = sF(S^{-1}(x, y, z))$

Deformacje

- Do dziedziny funkcji można zastosować również przekształcenia nieliniowe

- Zdeformować można również wynik działania funkcji

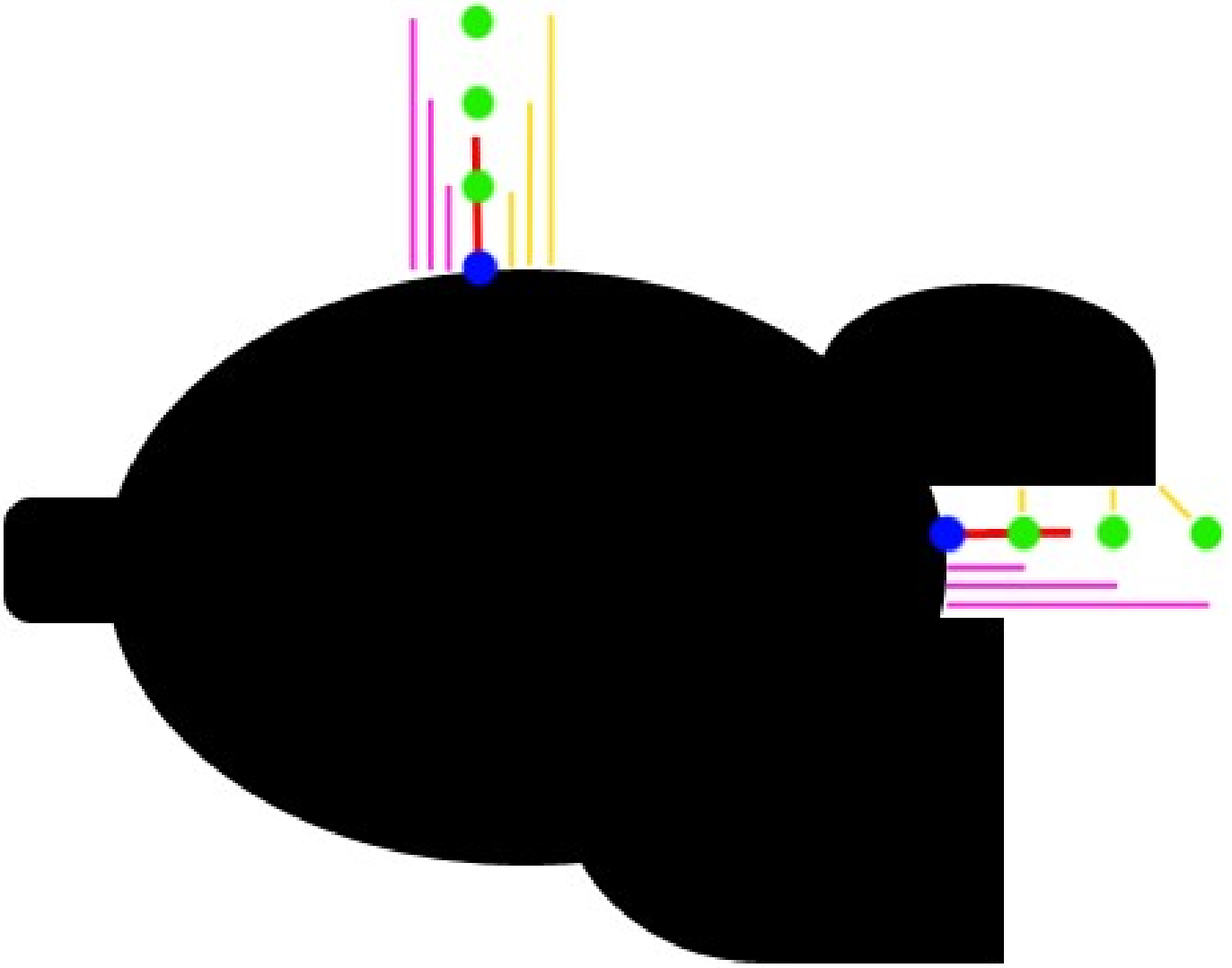



- W obu przypadkach funkcja wynikowa może nie być funkcją odległości

Zalety użycia funkcji odległości

- Zapewniają globalną informację o geometrii sceny
- Mały koszt uzyskania miękkich cieni i ambient occlusion
- Kilka ewaluacji funkcji odległości wzdłuż
  - promienia od punktu do światła – dla cieni
  - wektora normalnego powierzchni – dla AO
- Agregacja wyników – propozycje w "Rendering worlds with two triangles" I. Quileza
  http://www.iquilezles.org/www/material/nvscene2008/rwwtt.pdf

Ambient occlusion



Implementacja na GPU

- Full-screen quad / trójkąt
- Pixel shader:
  - Śledzenie promienia w scenie
  - Proceduralnie zdefiniowana funkcja odległości

Schemat pixel shadera

```
float3 rayOrigin, rayDirection; // punkt startowy i kierunek promienia
// generowanie promienia...

float3 currentPosition = rayOrigin;

while(distance(currentPosition, rayOrigin) < FAR_CLIP_PLANE)
{
    float dist = distanceFunction(currentPosition);
    currentPosition += rayDirection * dist;
    if(dist < EPS)
    {
        float3 normal = calcNormal(currentPosition);
        float3 pixelColor;
        // liczenie oswietlenia
        break;
    }
}

return float4(pixelColor, 1.0f);
```

```
float distanceFunction(float3 P)
{
    // odleglosc do kulki o promieniu 3 przesunietej 5 jednostek w gore osi Y
    float3 inputPos = P;
    inputPos.y -= 5;
    float distSphere = distanceSphere(inputPos, 3);

    // odleglosc od boxa o rozmiarach 3, 20, 3 i zaokragleniu 0.5,
    // do ktorego zastosowano deformacje - twist zmienny w czasie
    float3 inputPos = P;
    inputPos.xz = Rotate(inputPos.xz, P.y / 10 * sin(gTime * 4.0f));
    float distBox = distanceRoundedBox(inputPos, float4(3, 20, 3, 0.5f));

    // odleglosc do "zmieszania" dwoch obiektow zmieniajacego sie w czasie
    float distBlendBox = distanceRouBox(inputPos, float4(6, 3, 3, 0.5f));
    float distBlendTorus = distanceTorus(inputPos, float2(5,3));
    float blendFactor = (sin(gTime)) * 0.5f + 0.5f;
    float distBlend = blendFactor * distBlendTorus + (1 - blendFac) * distBlendBox;

    // polaczeniu wszystkich wynikow poprzez minimum (suma powierzchni)
    float distToAll = min(distSphere, distBox);
    distToAll = min(distToAll, distBlend);

    return distToAll;
}
```

Liczenie wektora normalnego

```
float3 calcNormal(float3 P)
{
    float3 normal;
    const float EPS = 0.03f;

    normal.x = distanceFunction(float3(P.x + EPS, P.y, P.z))
             - distanceFunction(float3(P.x - EPS, P.y, P.z));
    normal.y = distanceFunction(float3(P.x, P.y + EPS, P.z))
             - distanceFunction(float3(P.x, P.y - EPS, P.z));
    normal.z = distanceFunction(float3(P.x, P.y, P.z + EPS))
             - distanceFunction(float3(P.x, P.y, P.z - EPS));

    return normalize(normal);
}
```

Integracja z rasteryzacją

- Renderowanie siatek trójkątów i obiektów które nie zyskują na wykorzystaniu pól odległości
- Takie same ustawienia kamery, świateł
- Problem głębi, więcej niż jedno rozwiązanie:
  - Tekstura głębi tworzona w etapie sphere tracingu, używana podczas rasteryzacji
  - Tekstura głebi tworzona w etapie rasteryzacji, używana podczas sphere tracingu
  - Odległość punktu od oka
- Cienie

Cienie

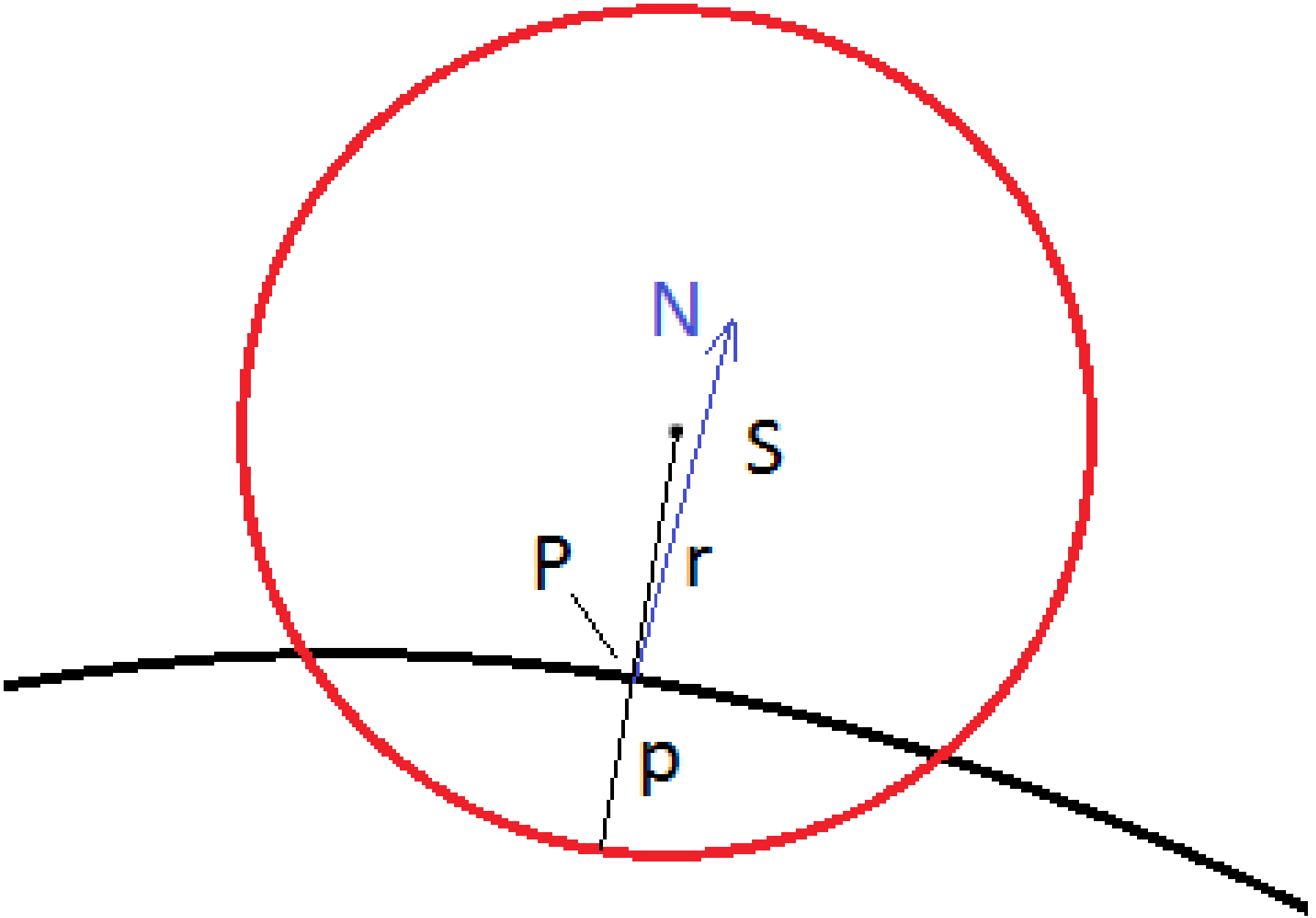
- Spójność wizualizacji
- Kilka możliwości rozszerzenia
- Wybrana: stworzenie mapy cienia w procesie sphere tracingu
  - Okrojony pixel shader (bez oświetlenia etc.)
  - Zapis odległości od światła do rendertargetu
  - Użycie przy rasteryzacji jak w klasycznym shadow mappingu

Kolizje i fizyka

- Prosta symulacja – tylko kulki z powierzchniami niejawnymi
- Funkcje odległości zapewniają naturalną informację do detekcji kolizji
- Aby znaleźć kolizję z kulą:
  - Ewaluować funkcję odległości w środku kuli
  - Sprawdzić czy wartość odległości jest mniejsza niż promień kuli

Kolizje i fizyka

- Wektor normalny kolizji
  - Wektor normalny powierzchni w punkcie kolizcji
  - Punkt kolizji nie jest znany
  - Przybliżenie – wektor normalny w środku kuli
- Rozwiązywanie kolizji
  - Odbicie prędkości względem wektora normalnego
  - "Wysunięcie" kuli z powierzchni o odległość *p* wzdłuż wektora normalnego (przydatne m.in. przy kolizjach spoczynkowych)


N
S
P
r
p

Symulacja w całości na GPU

- Dwa zestawy tekstur z aktualnymi danymi kulek w każdym pikselu
- W każdym zestawie dwie tekstury R32G32B32A32
  - W pierwszej: pozycja
  - W drugiej prędkość
- Shader odczytuje wartości z jednego zestawu tekstur i zapisuje do drugiego
  - Aktualizacja pozycji i prędkości (grawitacja)
  - Detekcja i rozwiązywanie kolizji

Schemat wykonania programu