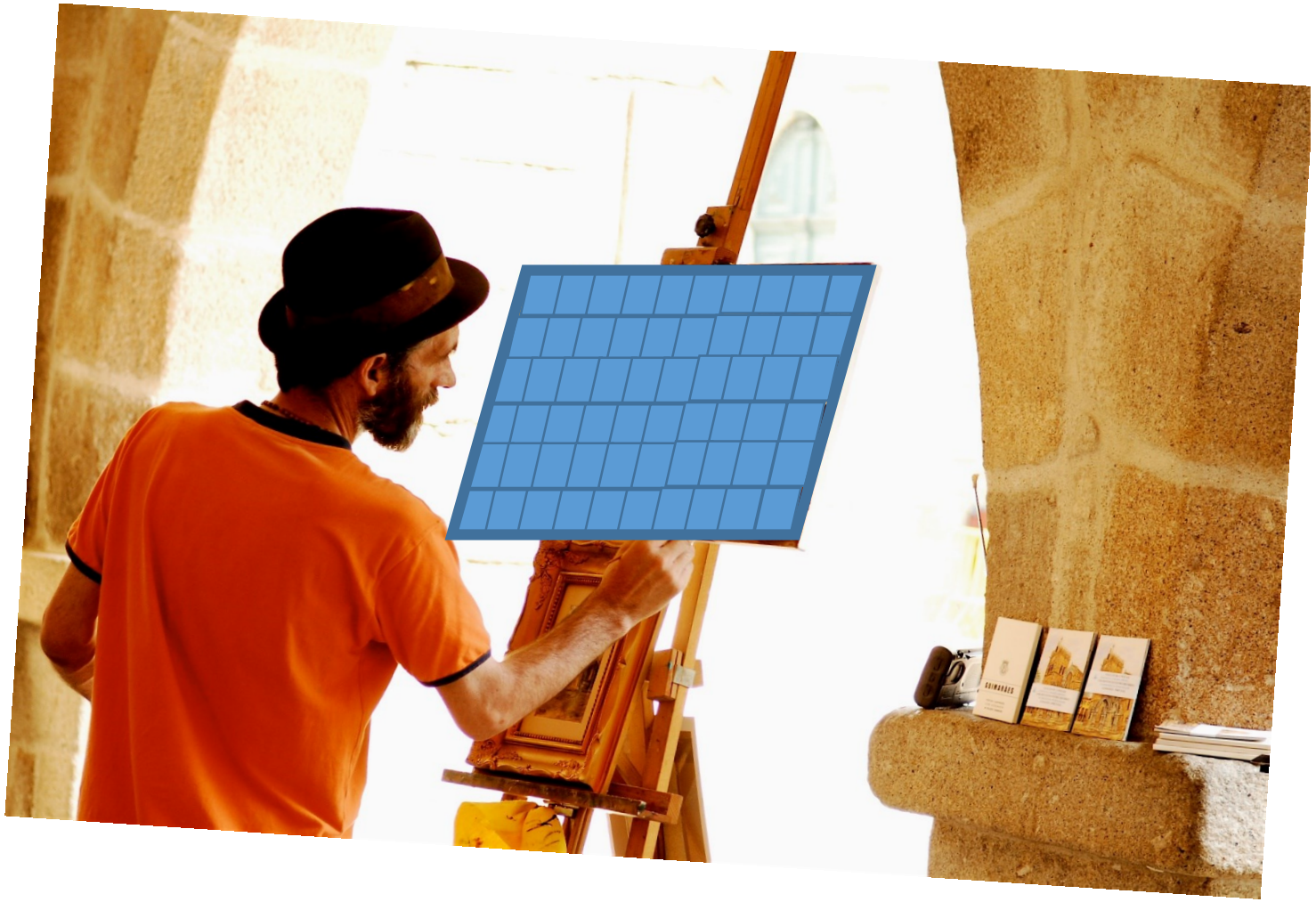
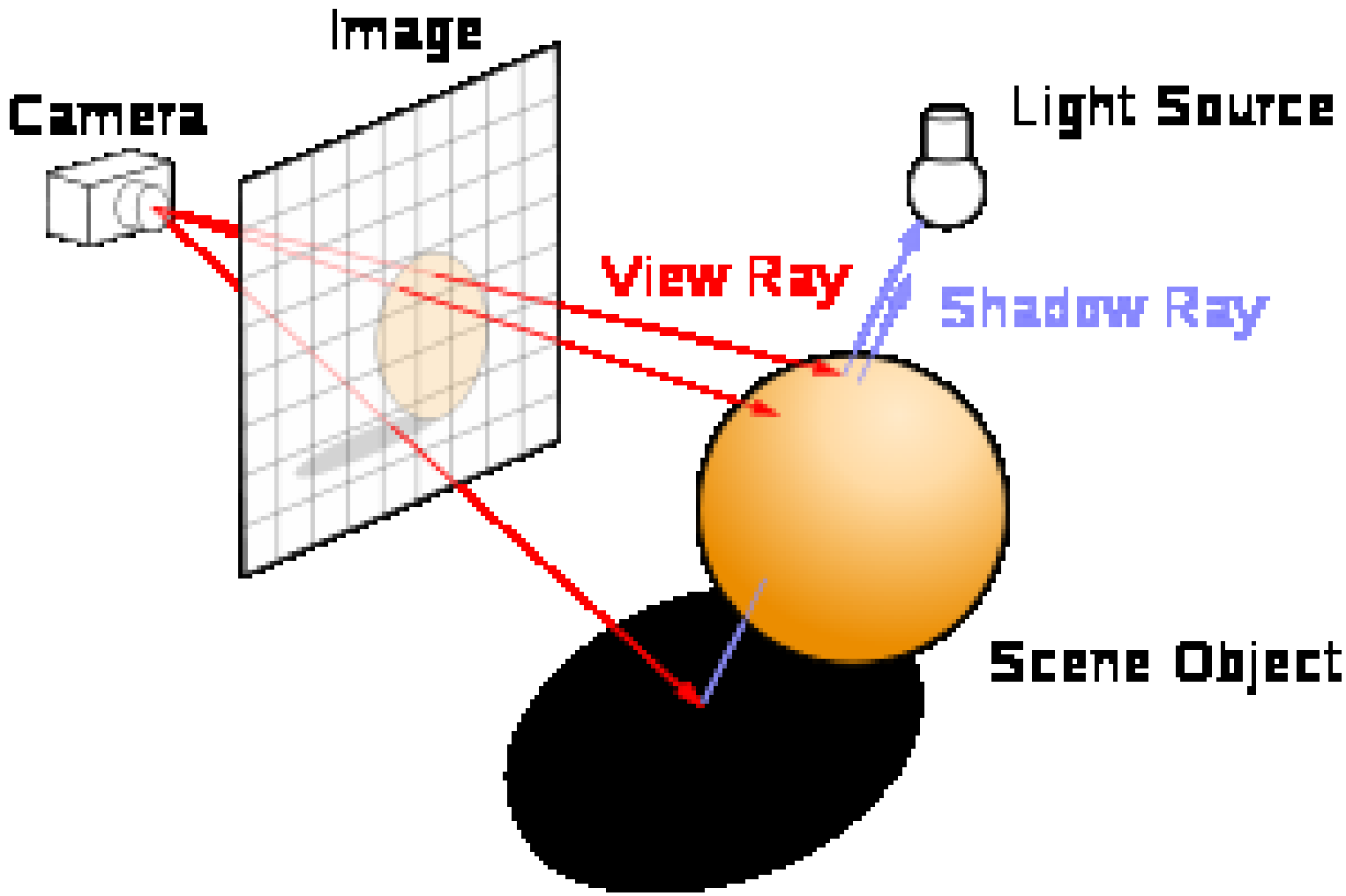
An Introduction to Implicit Modelling and Ray Tracing

What Is Ray Tracing (RT)?

What Is RT?

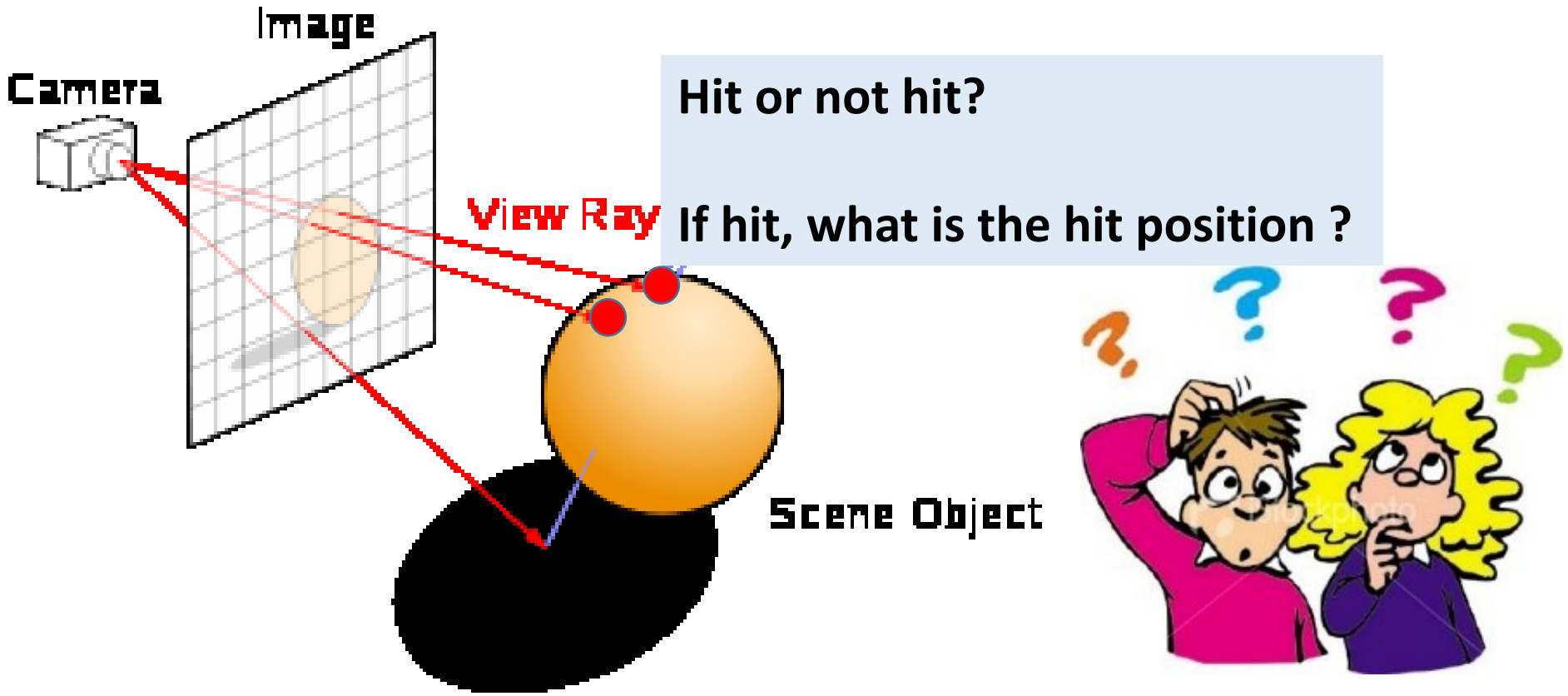


What Is RT?



Why Wasn’t RT Popular?

- Expensive to calculate ray-object intersections



Hit or not hit?

If hit, what is the hit position ?

Why Now?

- Powerful GPUs are available
- RT has been introduced as a new shader stage in DX12
- Real-time RT is now possible!

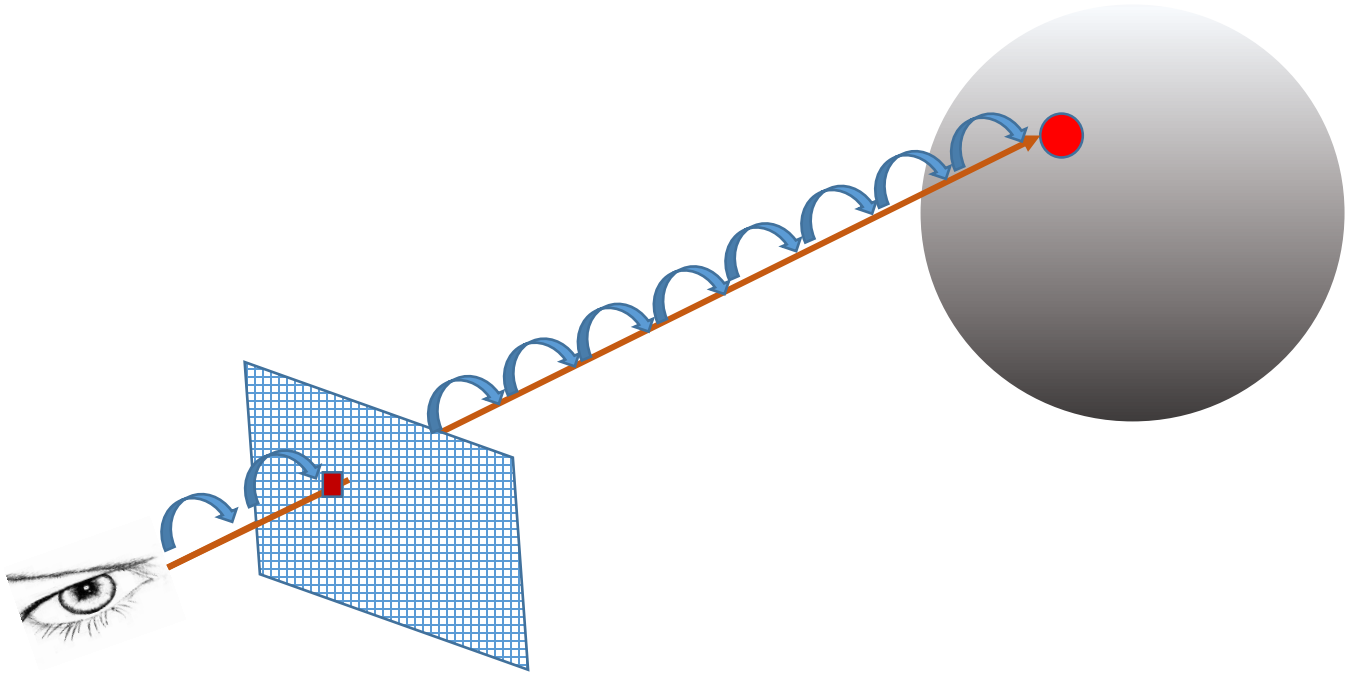
https://www.shadertoy.com/view/wsSXzz

How to find Ray-Object Intersection

- In general, a very tough task to find the exact solution
- Approximate solutions can be found with **ray marching**!
  - A numerical solution
  - Very efficient if an object is modelled in **distance functions**

- Distance functions?
  - What are they?
  - How can we represent a geometry as a function?

What Is Ray Marching



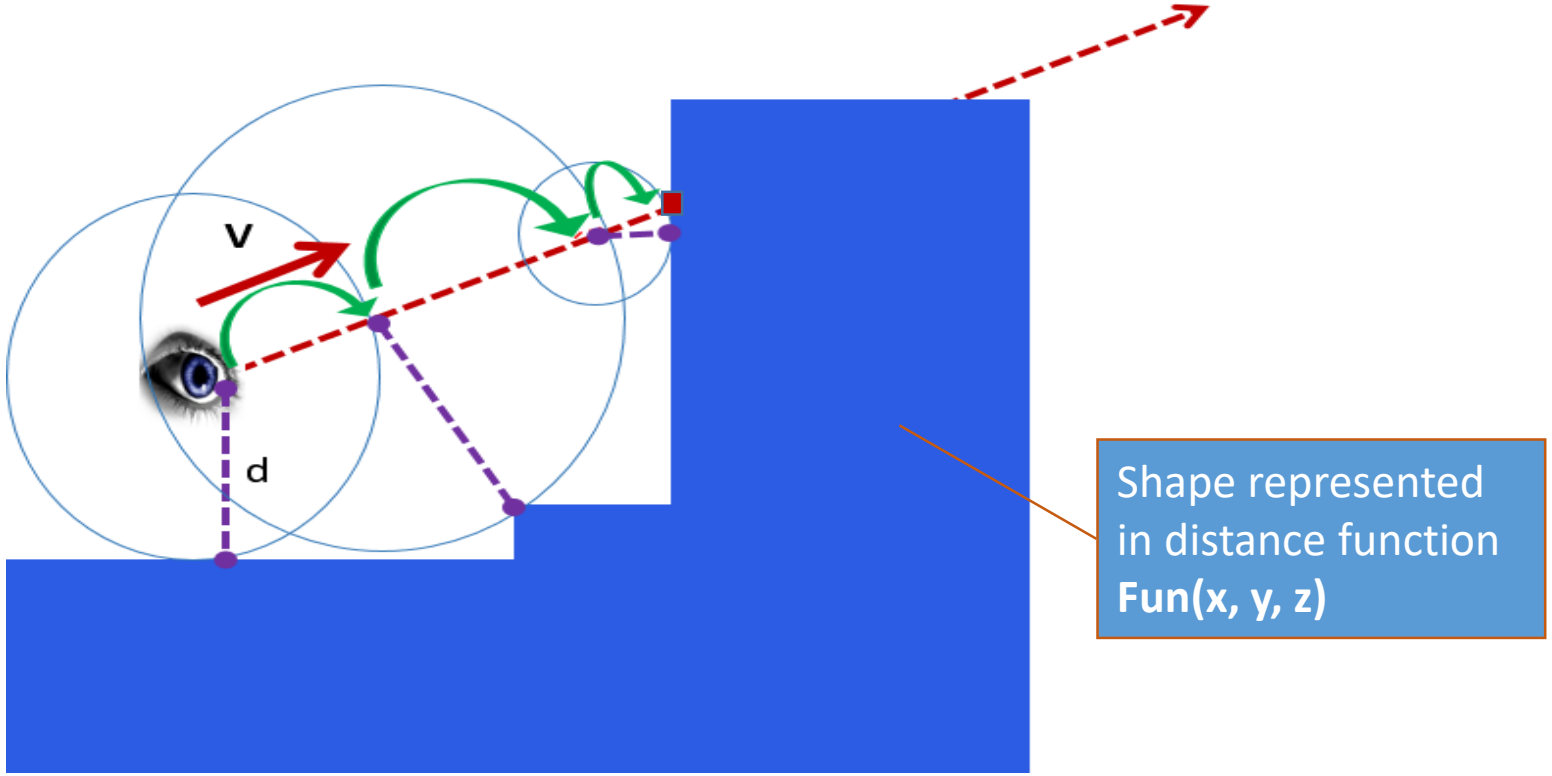
Ray marching: Distance function efficient

- The step sizes of ray marching can be directly obtained from the distance function:

NextP=currentP + d***V**

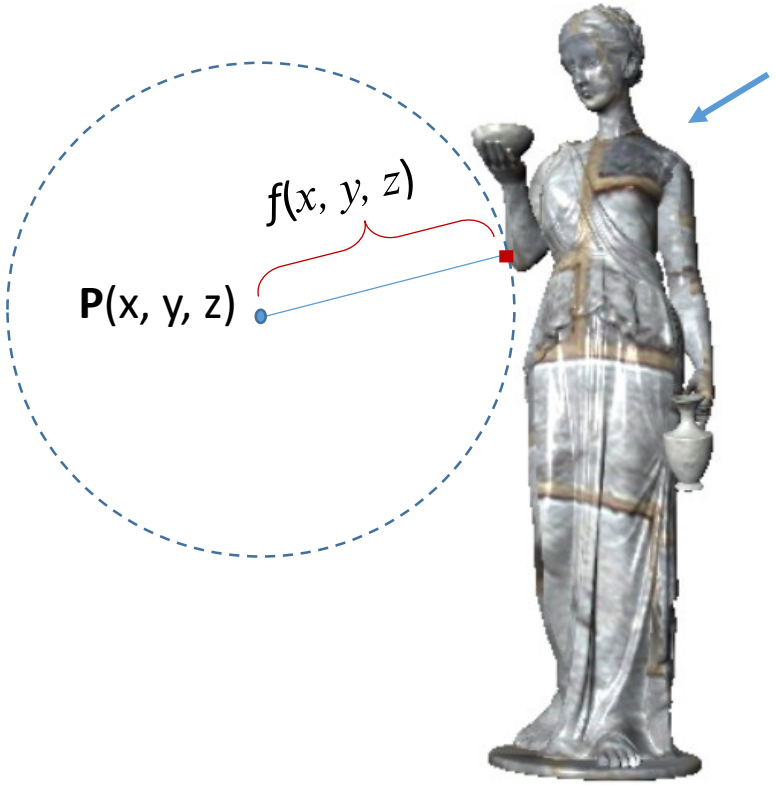
|V|=1

D=Fun(currentP)



What Is a Distance Function?

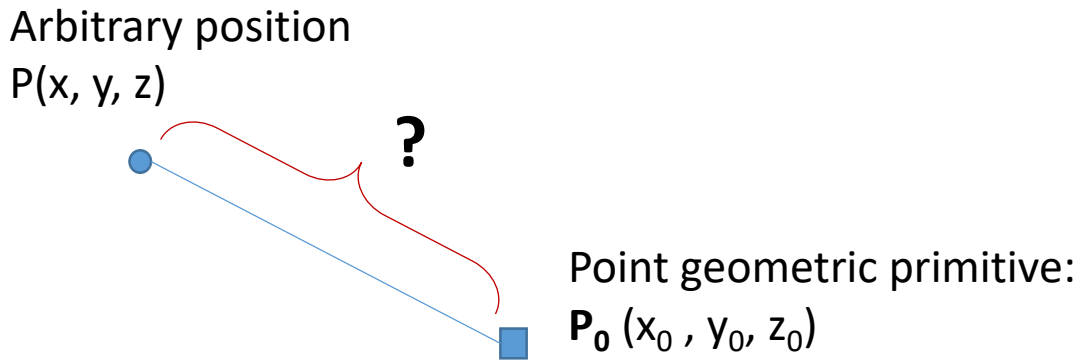
- A function of position in 3D, its value at **P**(x, y, z) represents the distance from the point **P** to the object
- But how to find the required distance function for a given shape?



What is the shortest distance from **P** to the object?

Thinking about geometric primitives!

- Point geometric primitive:



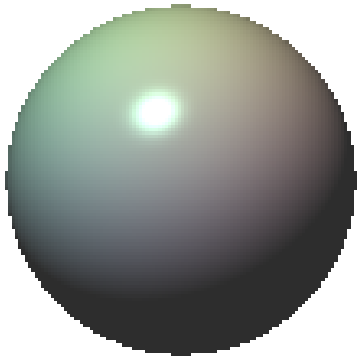
Dist(P, P0)=$\sqrt{(x-x_0)^2+(y-y_0)^2+(z-z_0)^2}$

Sphere: as a Distance Function to a Point

http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm

- GLSL code:

```
float sdSphere( vec3 P, vec3 C, float r )  //C: sphere centre; r: sphere radius
{
  return length(P-C)-r;
}
```



Distance to a Collection of Points!

- For a given set of points, S={$\mathbf{P}_0$, $\mathbf{P}_1$, …, $\mathbf{P}_n$}, the distance between a point **P** and S is defined as the minimum of the distances between **P** and the points $\mathbf{P}_0$, $\mathbf{P}_1$, …, $\mathbf{P}_n$:

$$\text{Dist}(\mathbf{P}, S)=\min\{dist(\mathbf{P}, \mathbf{P}_i): i = 0, 1, 2, \dots, n)$$

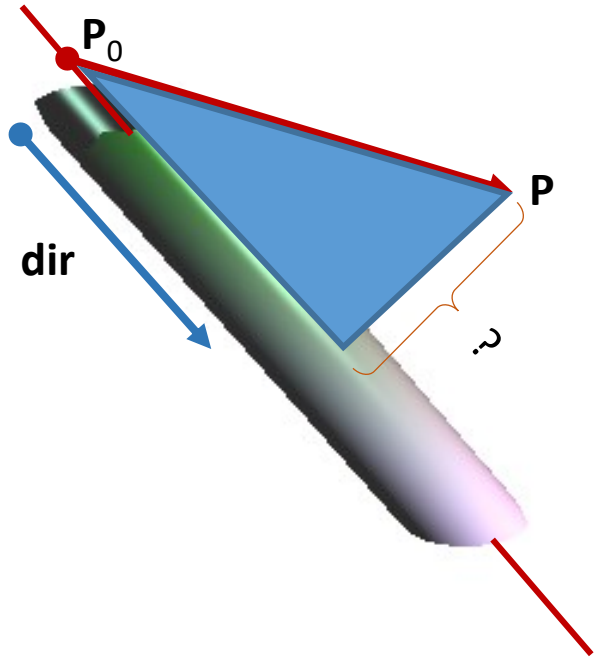
  - Its level set corresponding to a collection of spheres of the same size located at $\mathbf{P}_0$, $\mathbf{P}_1$, …, $\mathbf{P}_n$
- For example, the distance function of two points defines two spheres:

Cylinder: as Distance Function to a Line

- GLSL code:

```
float sdCylinder( vec3 P, vec3 P0, vec3 dir )
{
   vec3 V=normalize(dir);

  return length(P-P0 - dot(P-P0, V)*V);
}
```

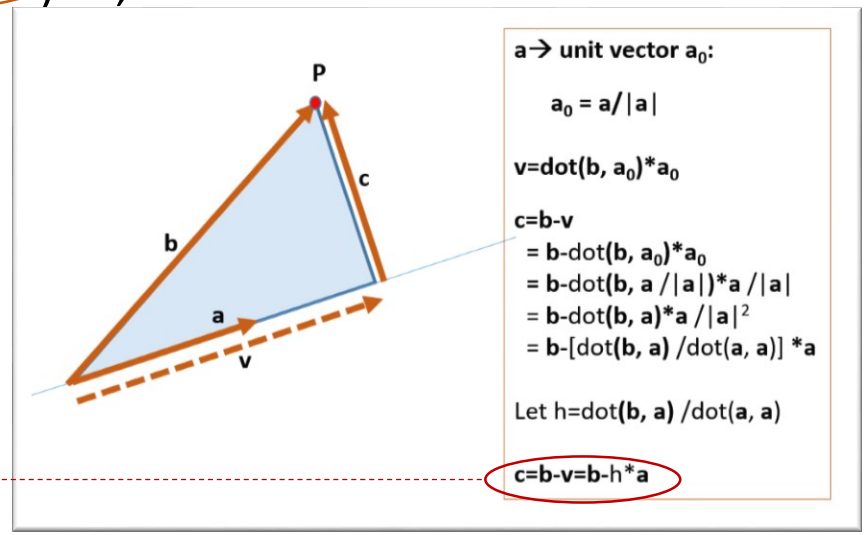


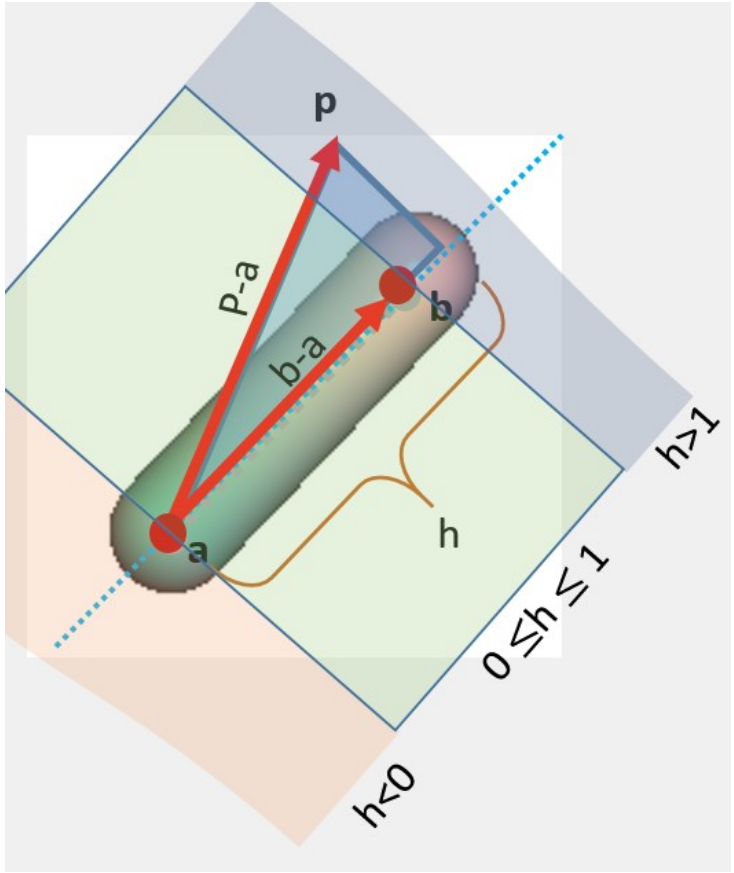
Capsule: as a Distance Function to a Line Segment

http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm

- GLSL code:

```
float sdCapsule( vec3 P, vec3 a, vec3 b, float r )
{
    vec3 pa = P - a, ba = b - a;
    float h = clamp( dot(pa,ba)/dot(ba,ba), 0.0, 1.0 );
    return length( pa - ba*h ) - r;
}
```



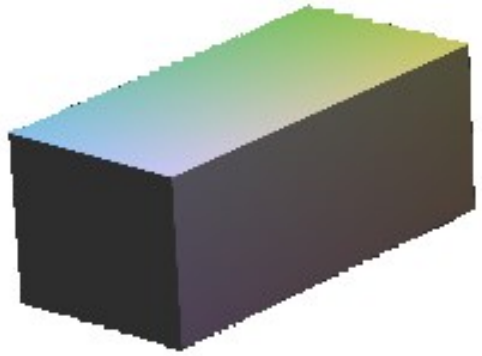


Box: as a Distance Function to Six Planes

http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm

- GLSL code:

```
float sdBox( vec3 P, vec3 C, vec3 b )
{
  vec3 d = abs(P-C) - b;
  return max(d.x, max(d.y,d.z) );
}
```

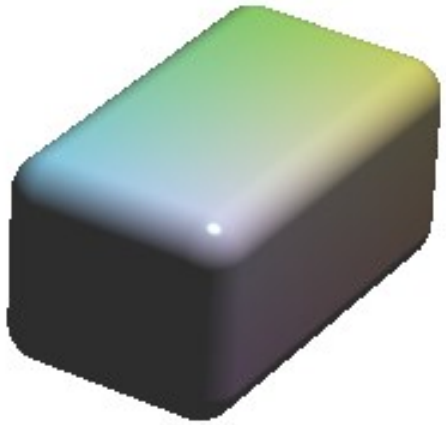


Round Box: as a Distance Function Six Planes

http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm

- GLSL code:

```
float udRoundBox( vec3 p, vec3 b, float r )
{
  return length(max(abs(p)-b,0.0))-r;
}
```

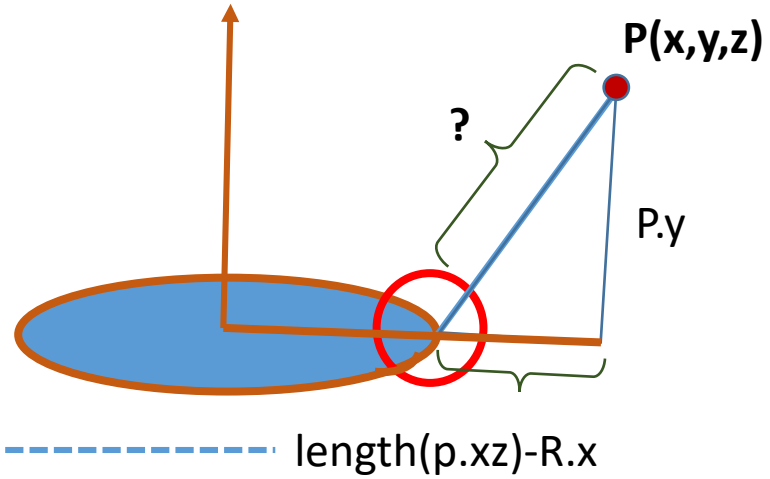


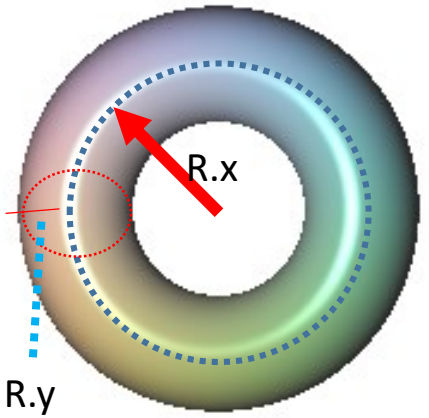
Torus: as a distance Function to a Circle

http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm

- GLSL code:

```
float sdTorus( vec3 p, vec2 R )
{
  vec2 q = vec2(length(p.xz)-R.x,  p.y);
  return length(q)-R.y;
}
```





Plane as a Distance Function

http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm

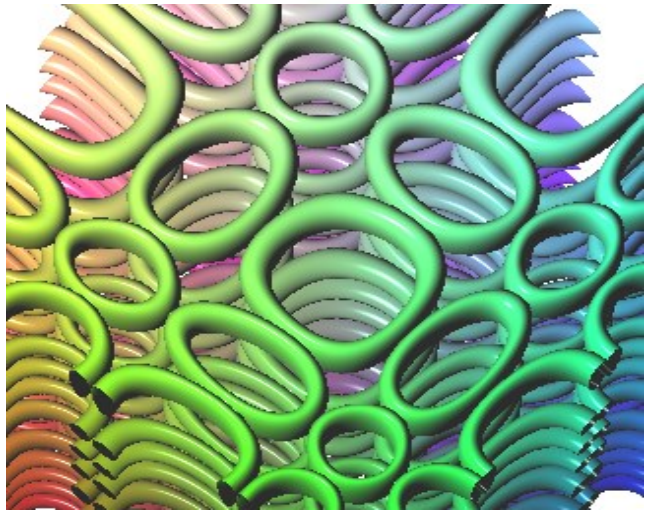
- GLSL code:

```
float sdPlane( vec3 P, vec3 Po, vec3 n )
{
  vec3 N1=normalize(n);
  return dot(P-P0,N1);
}
```

Implicit Shape Instancing

- Subdivide space into grids
- Translate grids centres to the coordinate origin
- Define implicit function in gridded space

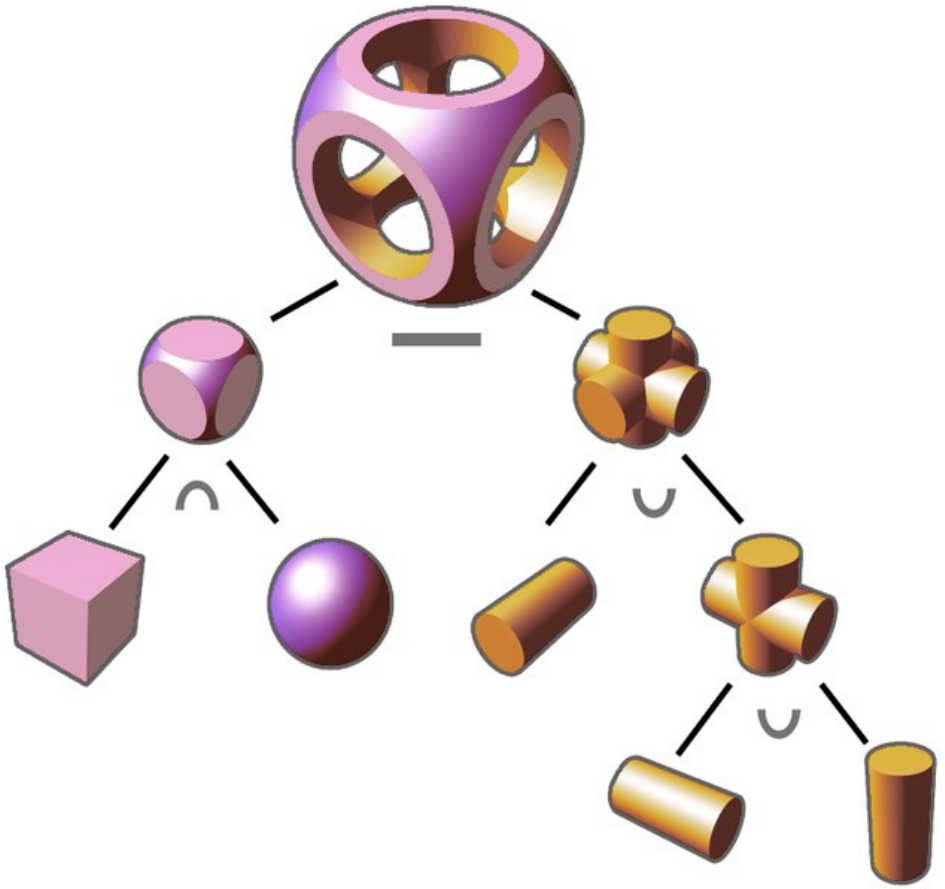
```
float impInstancing( vec3 p, vec3 c )
{
    vec3 q = mod(p,c)-0.5*c;
    return sdTorus( q, vec2(1, 0.2) );
}
```



How to construct complex shapes

- CSG
  - Blending simple geometric objects to construct a shape of interest
- Convert explicit objects into implicit ones
  - Mesh and parametric surface implicitization
  - Distance functions to points, curves, polygon or polyhedron
  - Data fitting and approximation
  - Accumulated field functions
    - Blobby objects
  - … …
- Implicit splines (QL&JT)

Constructive Solid Geometry(CSG)

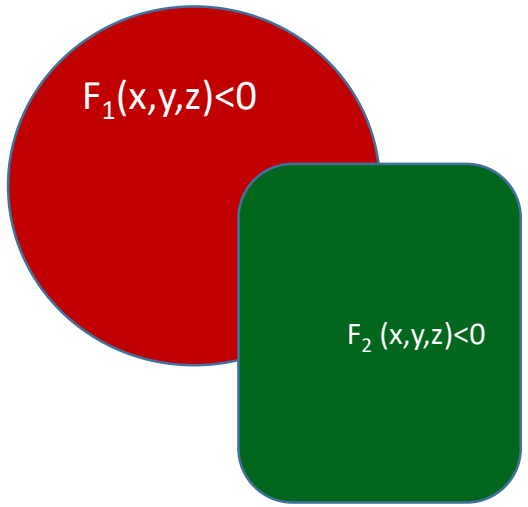


- Consider a shape as **a set** of points
- **Boolean operators** based
  - Complex shapes can be constructed from simple shapes using just set operations

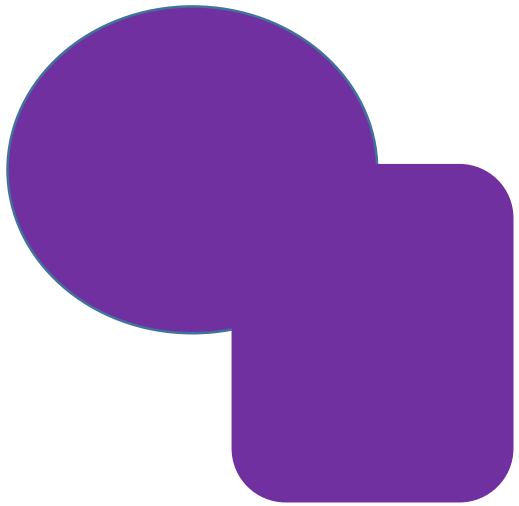
http://en.wikipedia.org/wiki/Constructive_solid_geometry

CSG: As Function Blending Operations

- Min-max operations

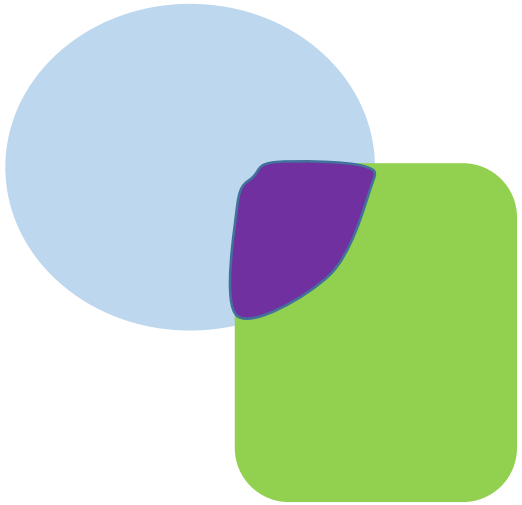


Union:
$F(x)=\min(F_1(P), F_2(P)) < 0$



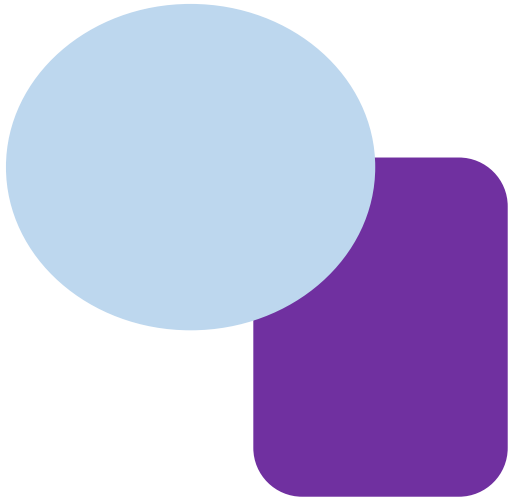
$F_1(x,y,z)<0$ **or**
$F_2(x,y,z)<0$

Intersection:
$F(x)=\max(F_1(P), F_2(P)) < 0$



$F_1(x,y,z)<0$ **and**
$F_2(x,y,z)<0$

Cut:
$F(x)=\max(-F_1(P), F_2(P)) < 0$



$F_1(x,y,z)\geq 0$ **and**
$F_2(x,y,z)<0$

Soft Blending

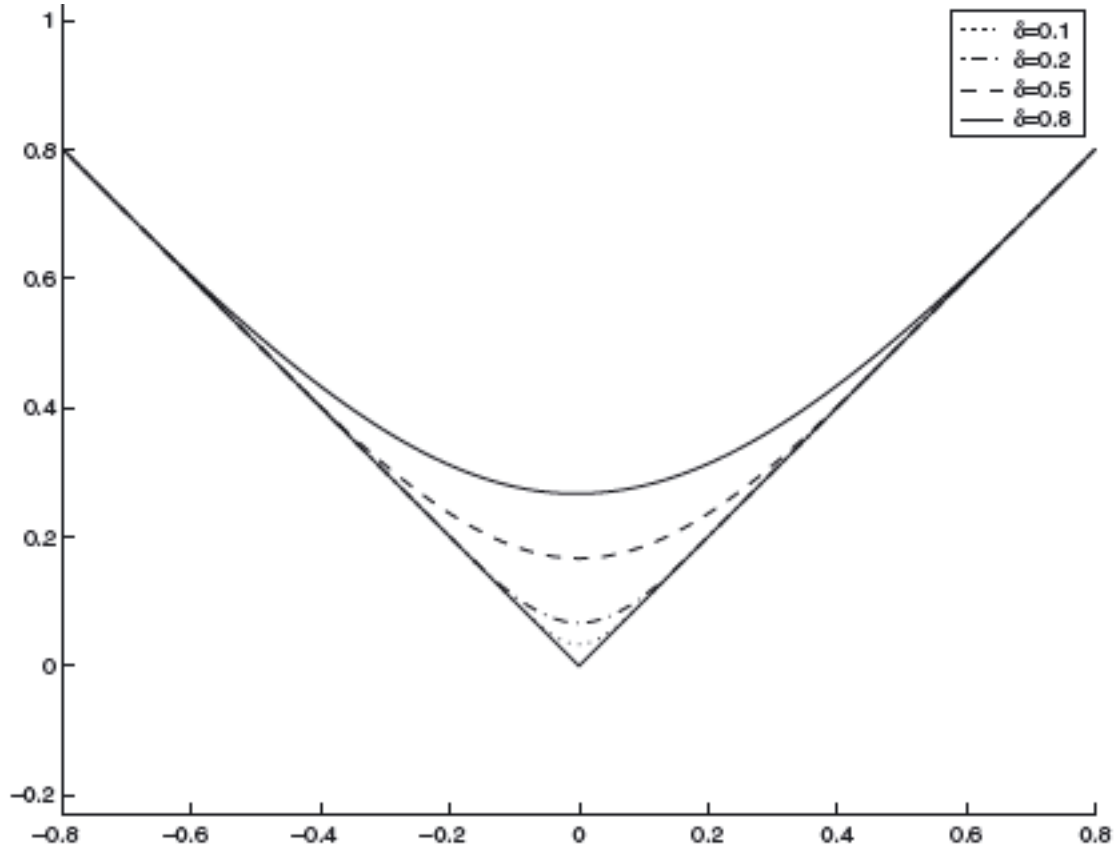
- Various methods
  - For example:

$$B(x,y) = x + y + \sqrt{x^2 + y^2}$$

$$B(x,y) = \frac{1}{\alpha}\log(e^{\alpha x} + e^{\alpha y})$$

- In general, most implicit shape blending operations are not shape preserving
- Soft implicit shape operations with blending range control can be developed by using soft absolute functions

Soft Absolute Functions (QL)



$$|x|_n = \frac{1}{(n+1)!2^n}\sum_{k=0}^{n-1}(-1)^k\binom{n-1}{k}G_n(x+n-2k-1),$$

$$G_n(x) = (x+1)^n|x+1| - (x-1)^n|x-1|$$

e.g. Quadratic soft absolution function:

$$|x|_2 = \begin{cases} |x|, & |x| > 2 \\ \frac{x^2}{2}\left(1 - \frac{1}{6}|x| + \frac{2}{3}\right), & |x| \le 2 \end{cases}$$

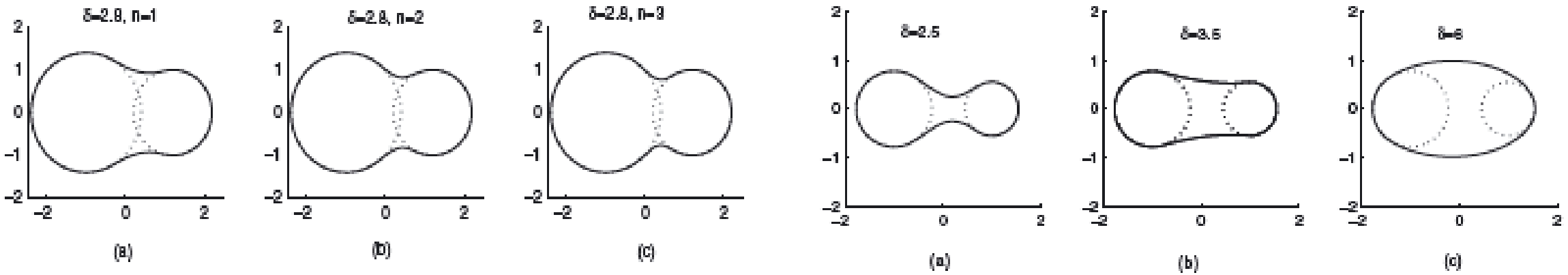
Blending range can be controlled by Introducing a number **δ**:

$$|x|_{n,\delta} = \frac{\delta}{n}\left|\frac{nx}{\delta}\right|_n$$

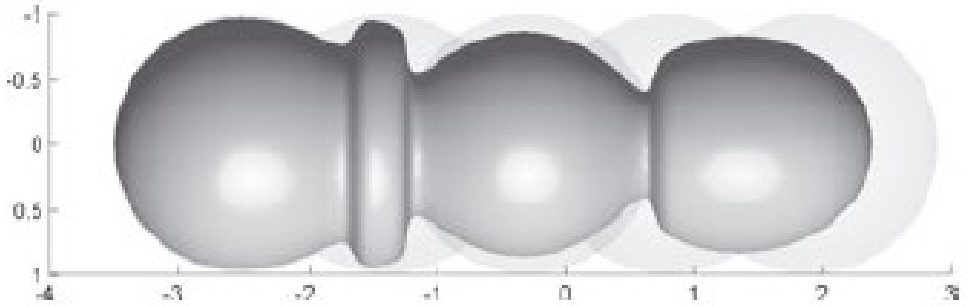
Shape-preseving Blending Operation

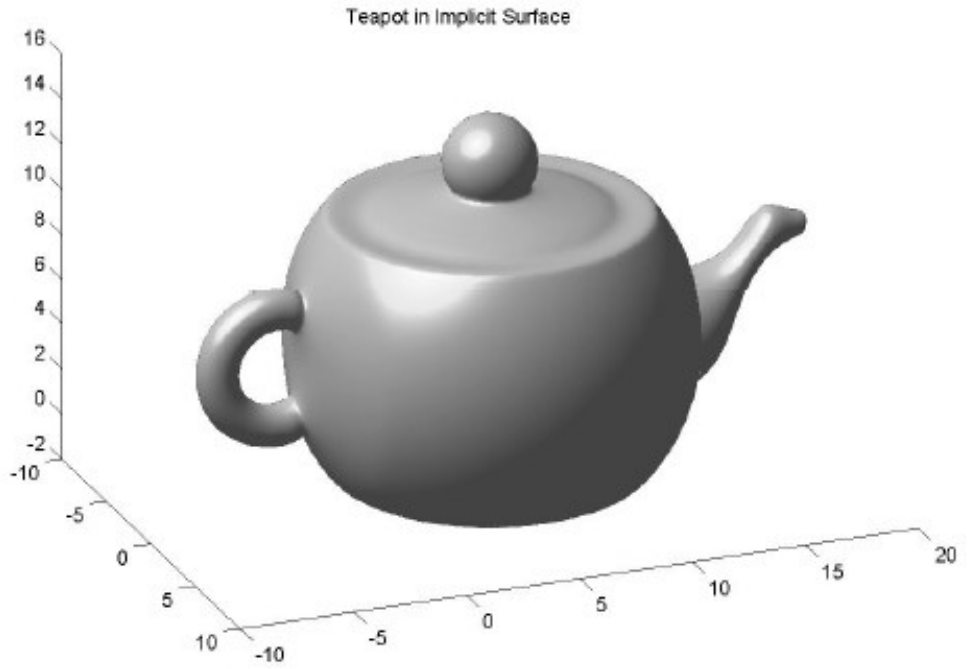
- Two implicit functions can be blended using

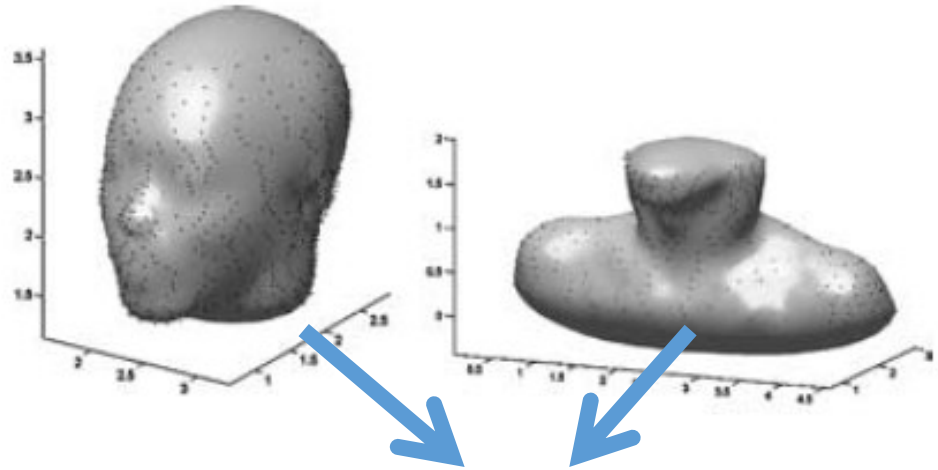
$$max_{n,\delta}(x, y) = \frac{1}{2}(x + y + |x - y|_{n,\delta})$$

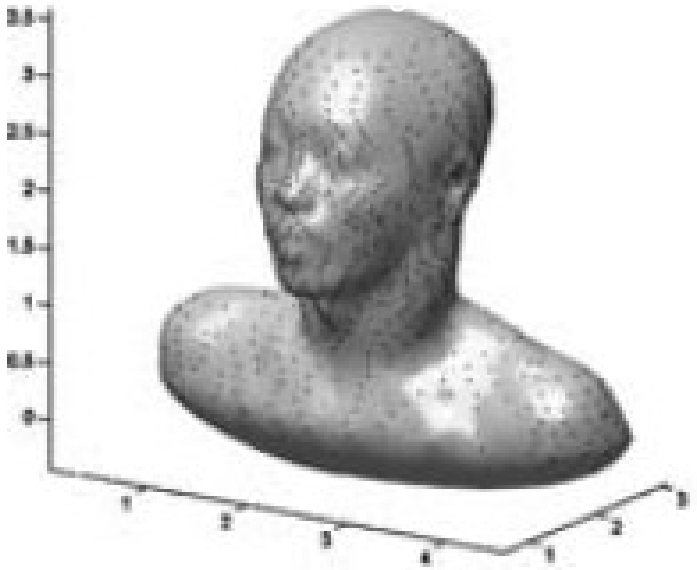


Soft Blending: Shape Preserving









Questions?