COMP3421

Splines, Extension Material

Robert Clifton-Everest

Email: robertce@cse.unsw.edu.au

Assignment 2 demos

- Monday week 11 (12th August)
  - 11am-2pm
  - You'll pick a 10 minute slot (I'll announce when you can do that)
  - You'll demonstrate to me or one of the tutors.
  - There are additional slots on Tuesday 13th 4pm-6pm for those that can't attend on the Monday.

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 $P_0 \ldots P_m$ 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

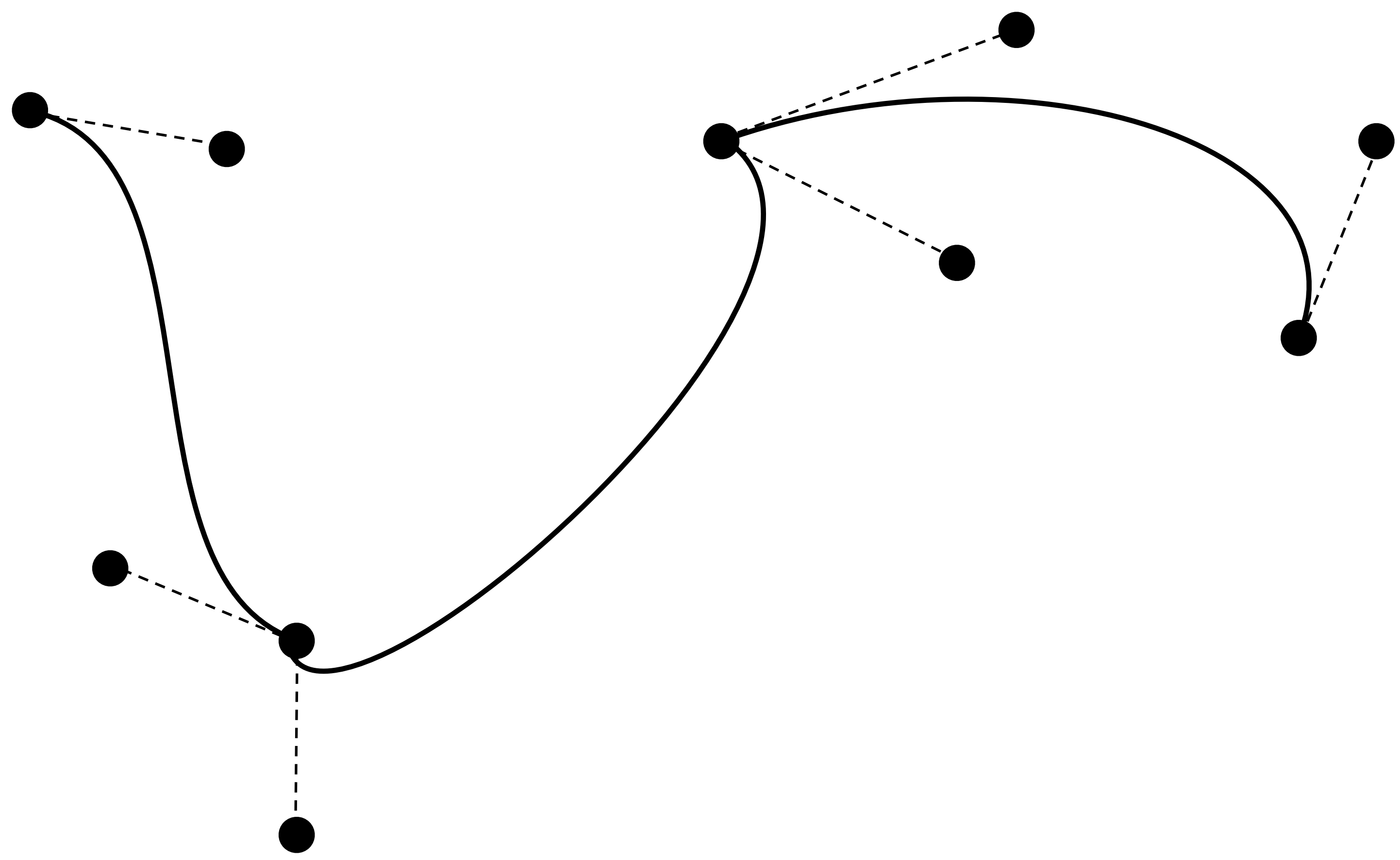
- A spline is a smooth piecewise-polynomial function (for some measurement of smoothness).
- The places where the polynomials join are called knots.
- A joined sequence of Bézier curves is an example of a spline.

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 *nth 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.
C^2: 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

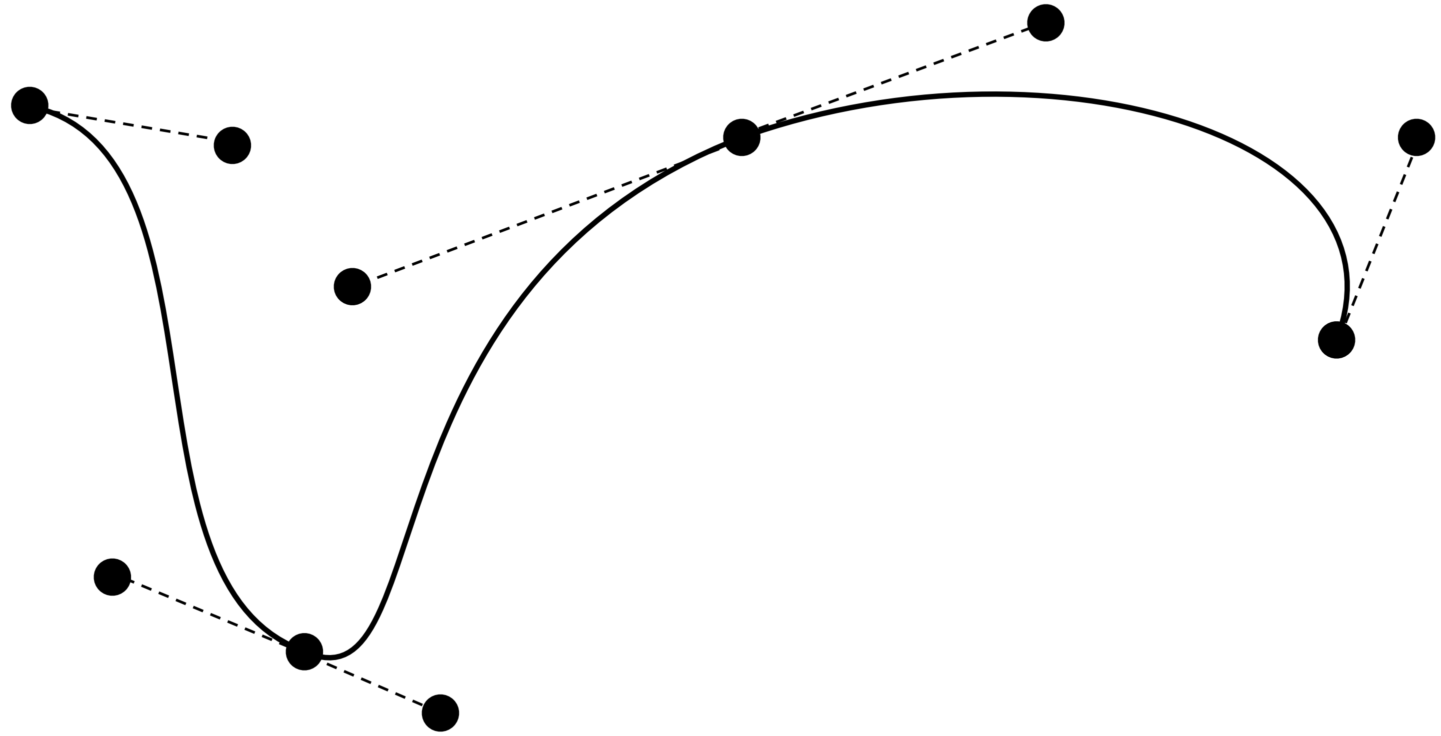
G^2 means the curve has continuous curvature.

Continuity

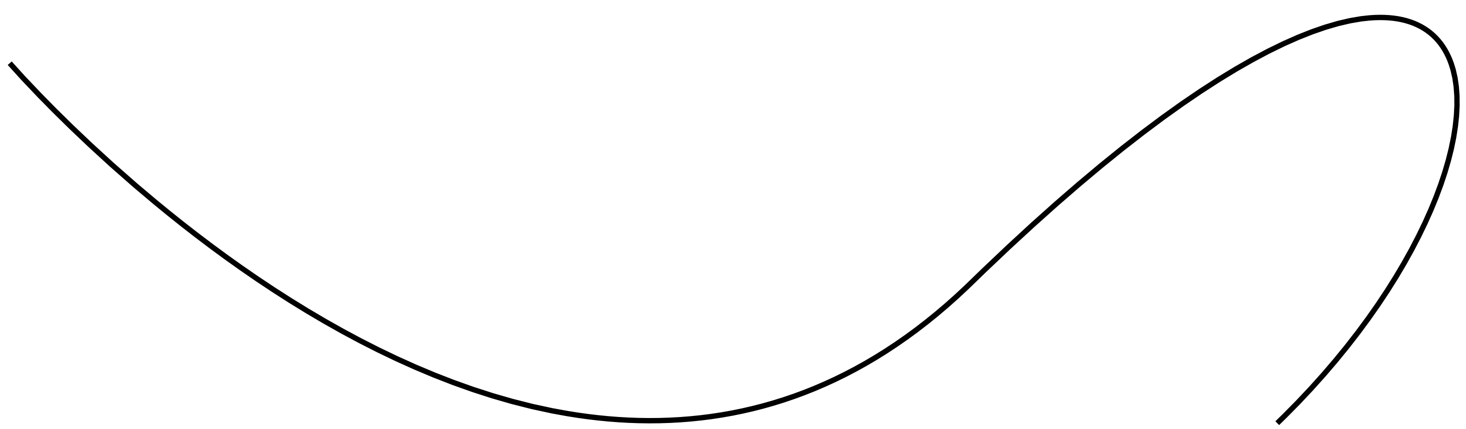
- Geometric continuity is important if we are drawing a curve.
- Parametric continuity is important if we are using a curve as a guide for motion.

Bézier splines

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

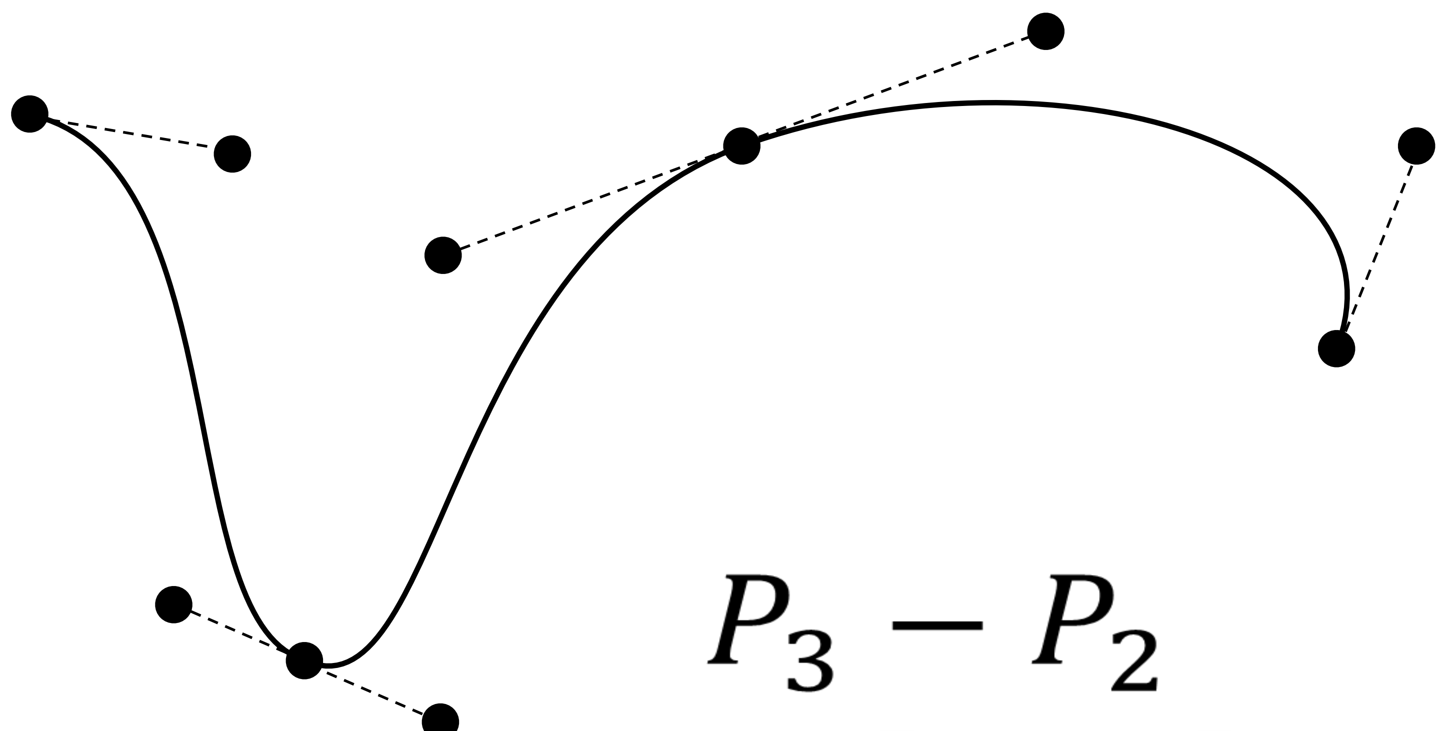


Drawing



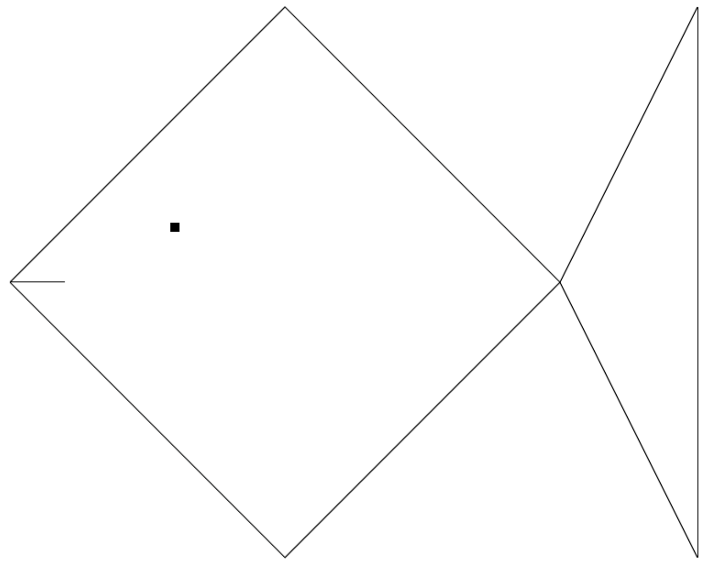
Bézier splines

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



$$P_3 - P_2 = P_4 - P_3$$

Motion



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:

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

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

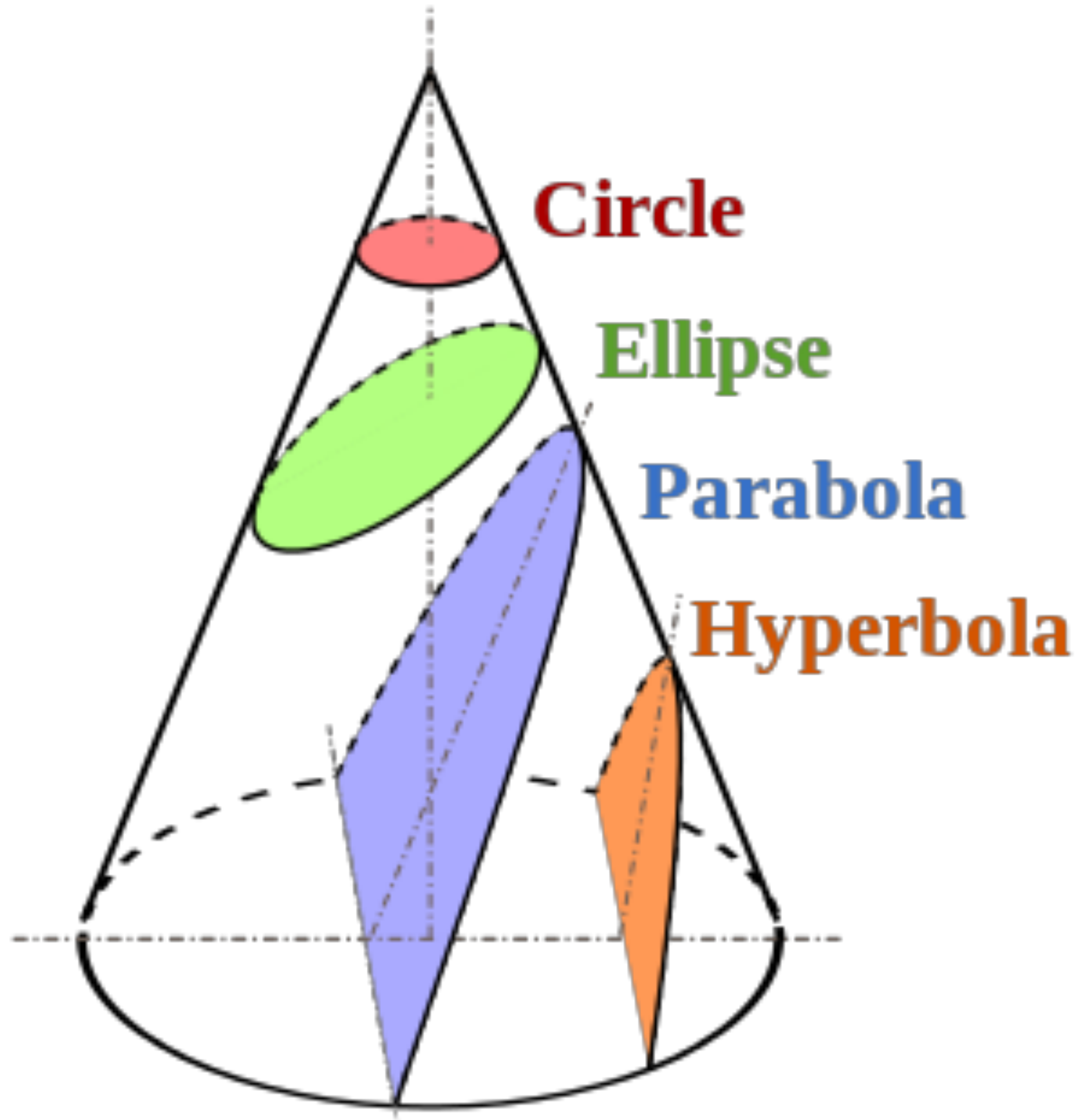
- The most commonly used B-splines are quadratic (m=2) and cubic (m=3).
- Uniform quadratic splines have C^1 (and G^1) continuity.
- Uniform cubic splines have C^2 (and G^2) continuity.

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 9 control points will look like this (0,0,0,0.25,0.25,0.5,0.5,0.75,0.75,1,1,1).

Incomplete

- Conic sections are the intersection between cones and planes.



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

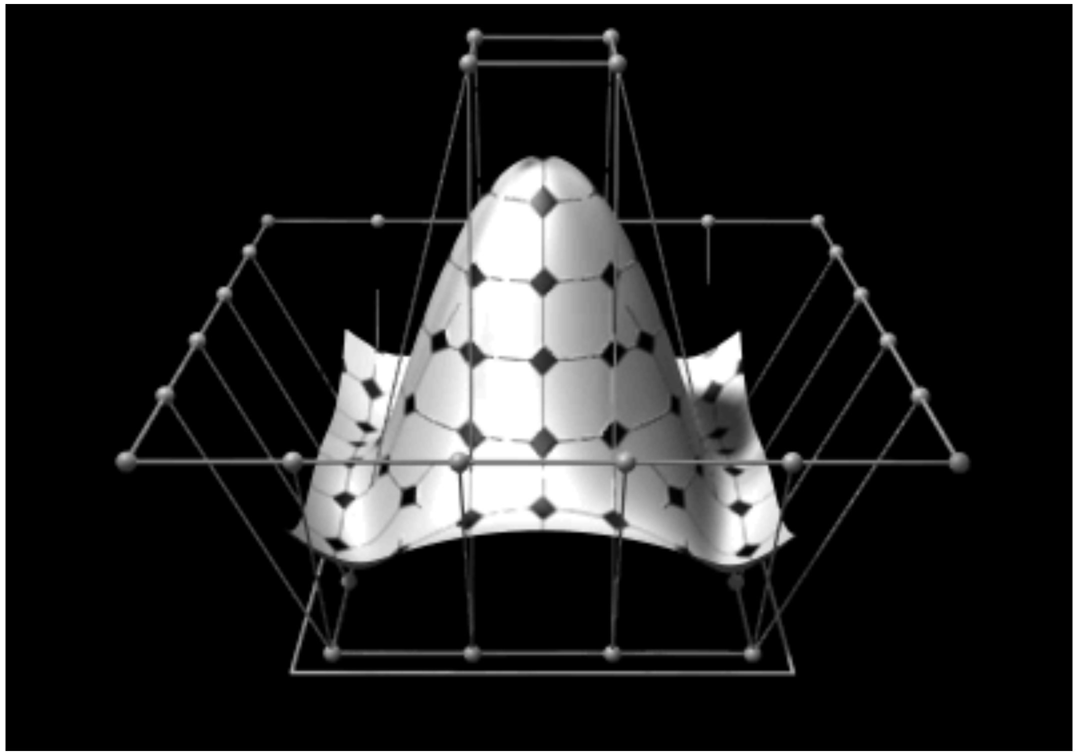
- We can also weight control points in B-splines to get rational B-splines:

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

NURBS

- Non-uniform rational B-splines are known as NURBS.
- NURBS provide a power yet efficient and designer-friendly class of curves.

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.

Extension

Modelling

- We’ve covered various modelling techniques in this course.
- Extrusion and surfaces of revolution are good for generating 3D shapes from 2D outlines.
- Beziers and NURBs give a user friendly way of defining curved surfaces.
- These techniques are good as it is trivial to tessellate them into polygons.

Implicit forms

- Implicit forms are good for ray tracing as we can calculate ray intersections easily. They are not, however, particularly user friendly.

Other modelling techniques

- We will look at some other modelling techniques.
- **Constructive Solid Geometry** is a very intuitive and compositional method of modelling.
- **Signed distance functions** have some nice properties and give rise to a powerful ray tracing technique.
- **Voxels** are a very simple idea but carry some performance caveats.

CSG

- Constructive solid geometry (CSG) builds up complex objects by composing together simple primitives.

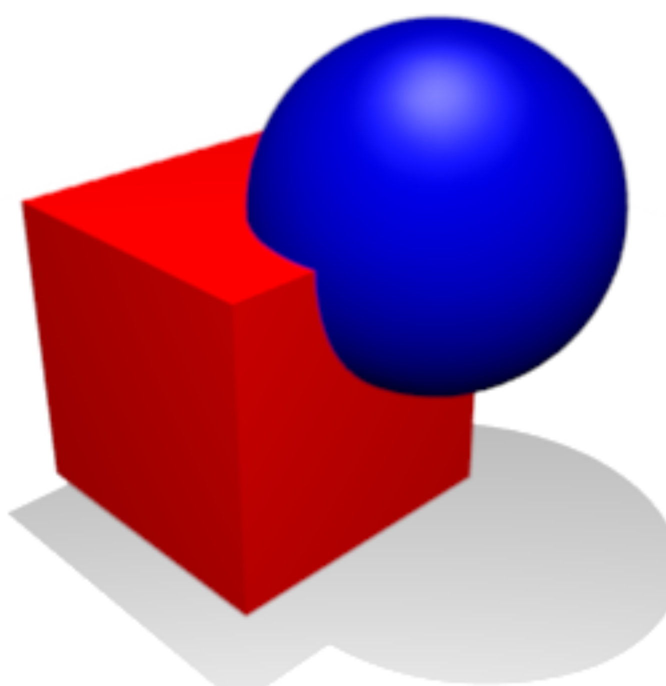
CSG

- The union of two objects is simply merging them together into one.

$$r = \text{... a red cube ...}$$

$$b = \text{... an offset blue sphere ...}$$

$$r \bigcup b =$$

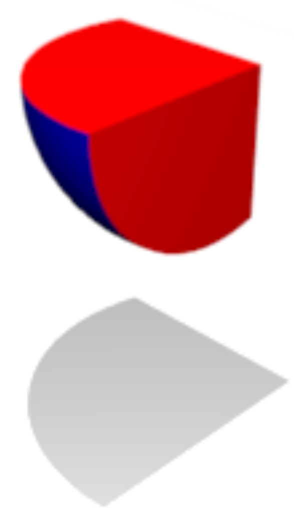


CSG

- The intersection gives the portion of space common to both objects.

$r = \text{... a red cube ...}$

$b = \text{... an offset blue sphere ...}$

$r \cap b =$ 

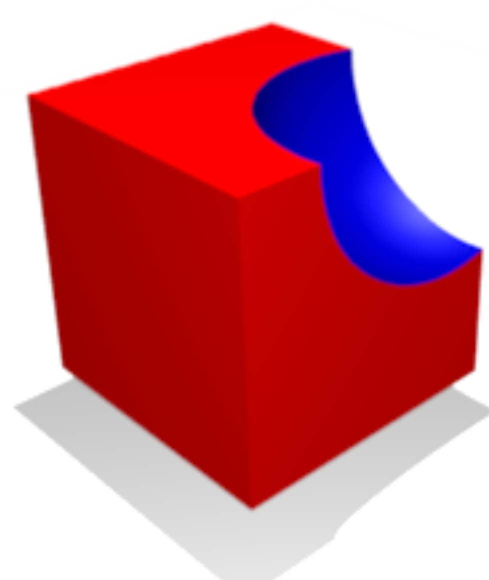
CSG

- It is also possible to subtract one object from another.

$$r = \text{... a red cube ...}$$

$$b = \text{... an offset blue sphere ...}$$

$$r - b =$$



CSG

- Try it yourself
- https://evanw.github.io/csg.js/

CSG

- If we have to objects in implicit form

$$F(P) = 0$$

$$G(P) = 0$$

- Then we can compute the union like so

$$F(P) = 0 \vee G(P) = 0$$

CSG

- Intersection poses a problem. Remember that all we have are the equations:

$$F(P) = 0$$
$$G(P) = 0$$

- We don’t know when a point is **inside** an object. We need something stronger than just an equation.

Signed distance functions

- A signed distanced function gives the distance from a given point to the closest point on the surface of the object.

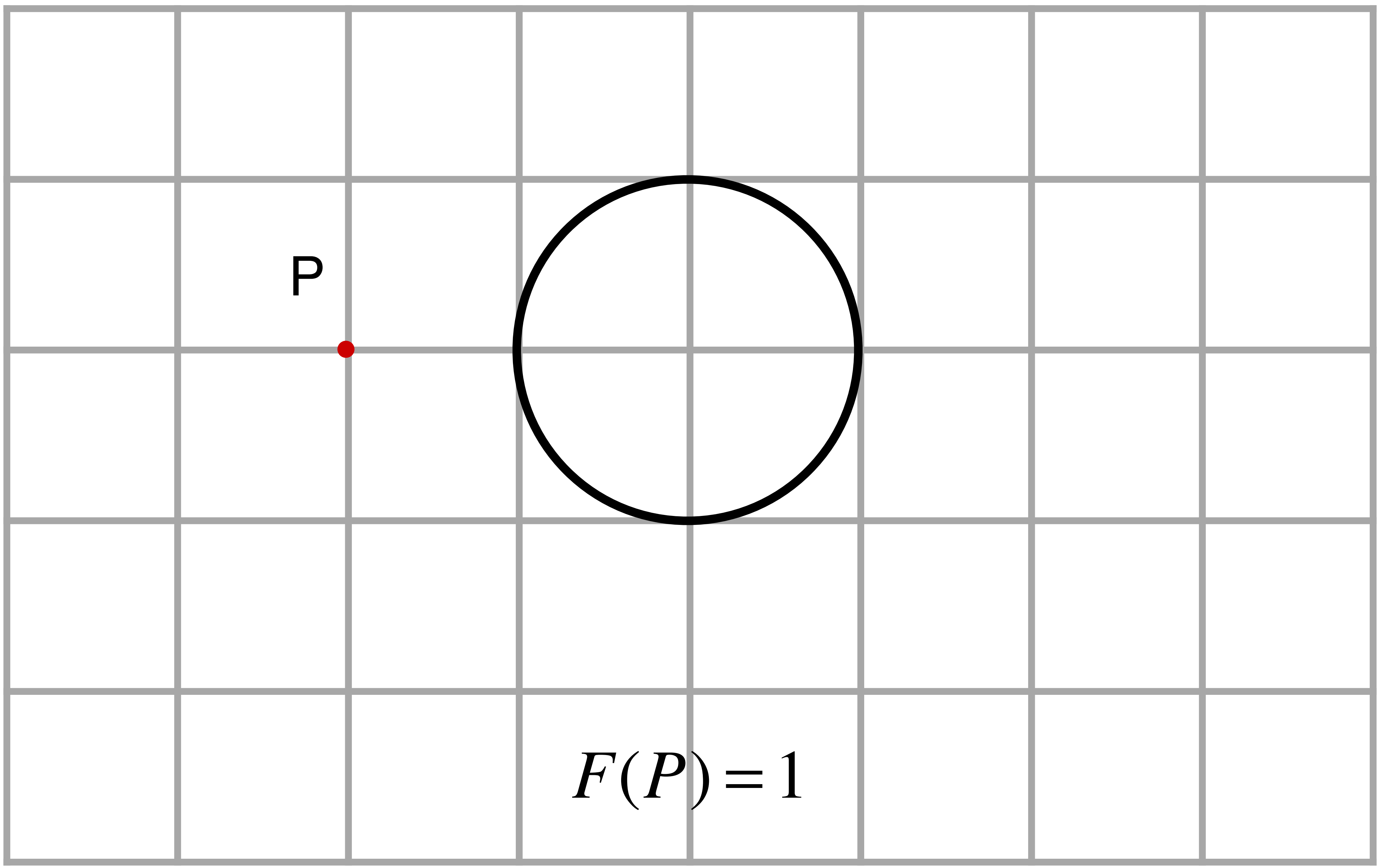
$$F(P) = \text{...the distance from } P \text{ to the object...}$$

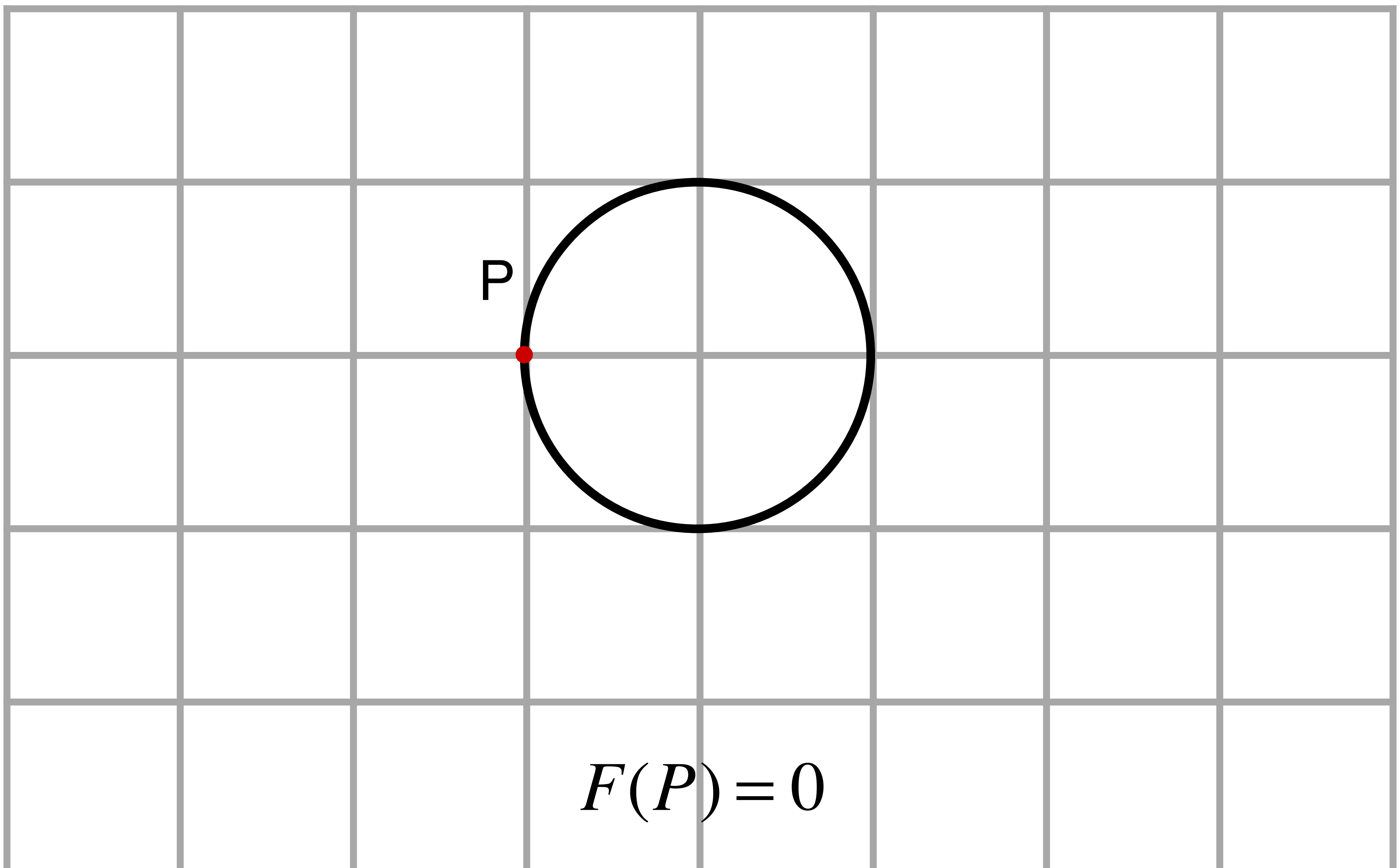
- If gives a negative distance when the point is inside the object.

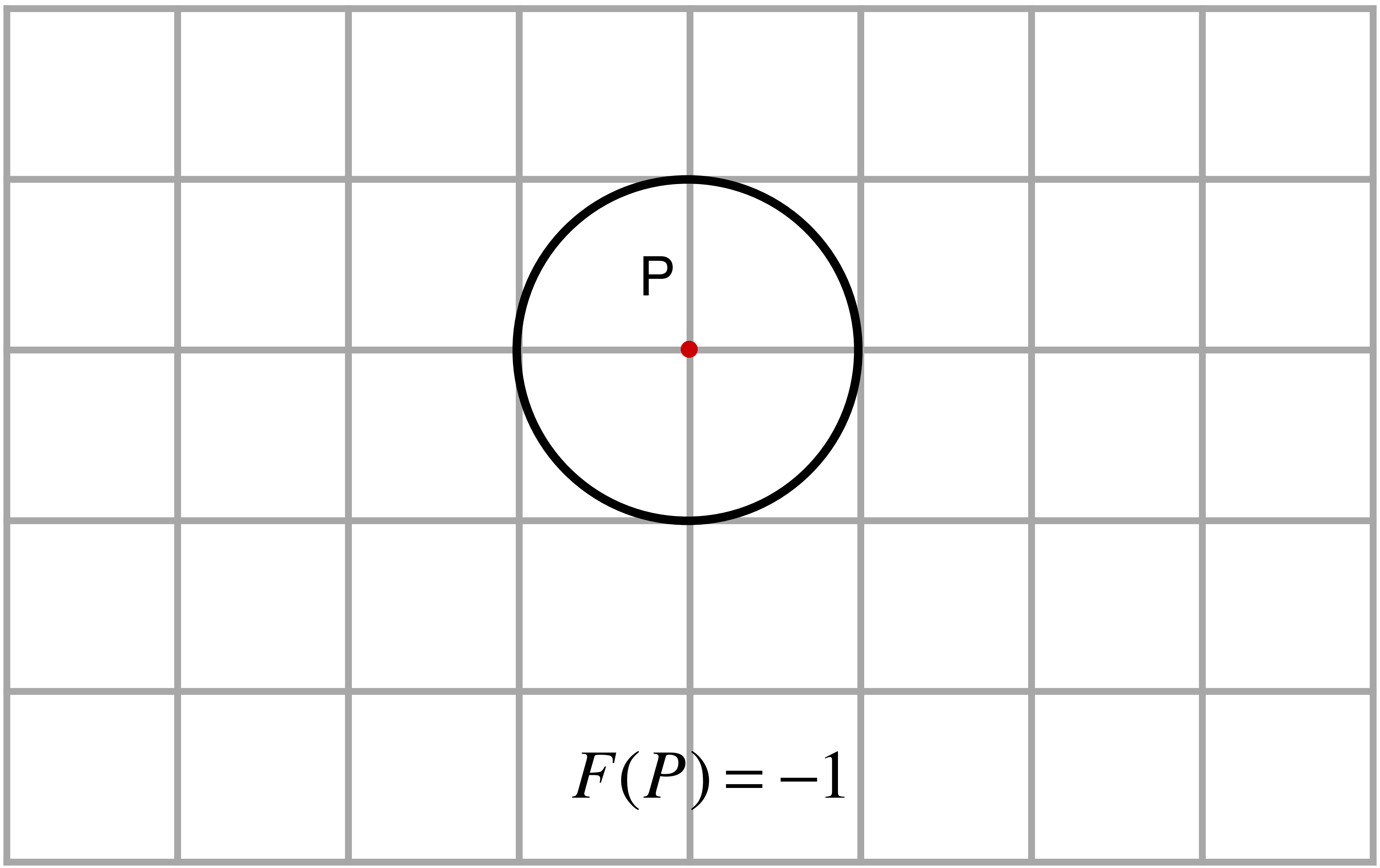
Signed distance functions

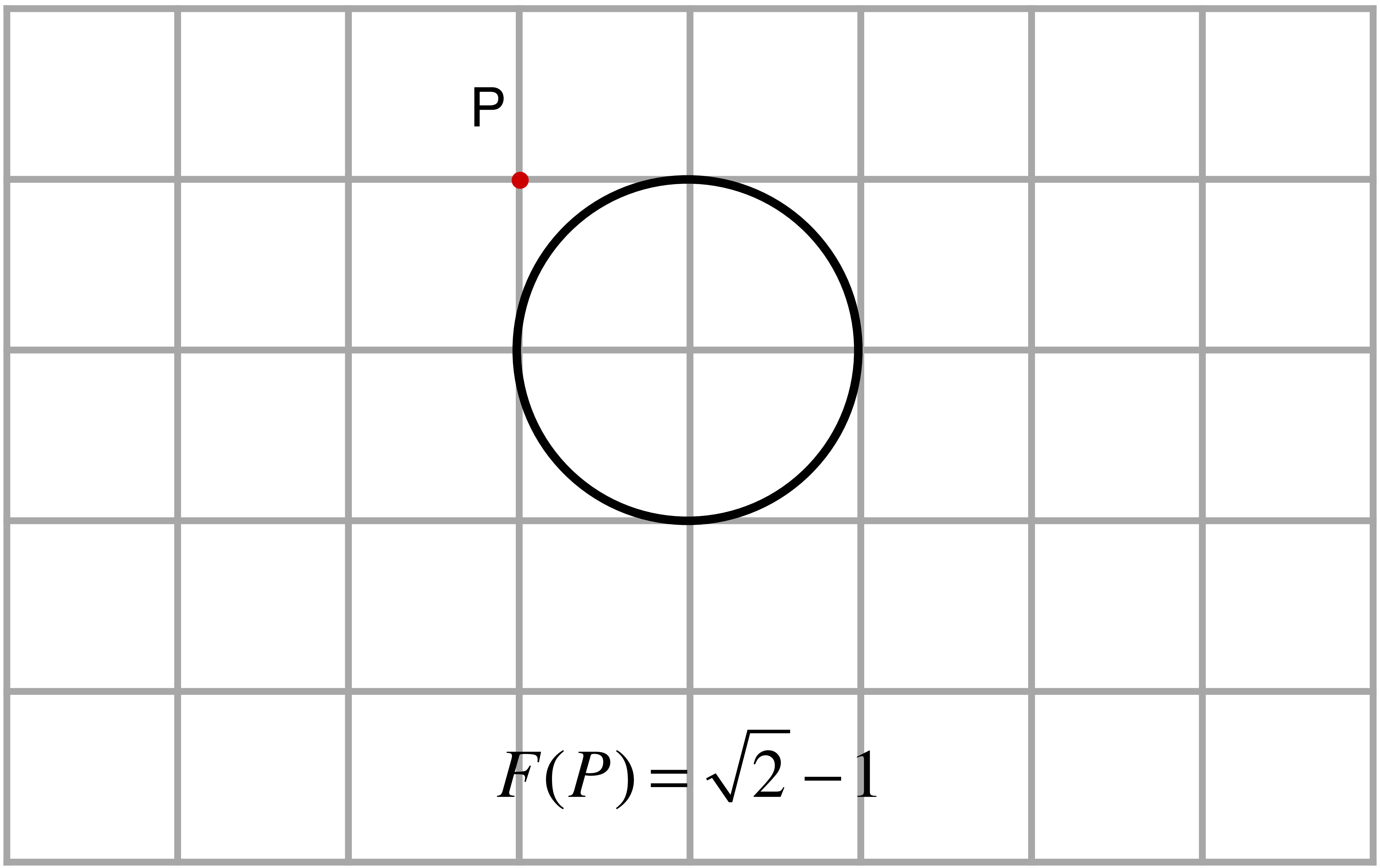
- For example, a unit circle has a signed distance function of

$$F(P) = P_x^{\,2} + P_y^{\,2} - 1$$


P
$F(P) = 1$


P
$F(P) = 0$


P
$F(P) = -1$


P
$F(P) = \sqrt{2} - 1$

Signed distance functions

- If our objects are signed distance functions then we can calculate the CSG operations as follows:

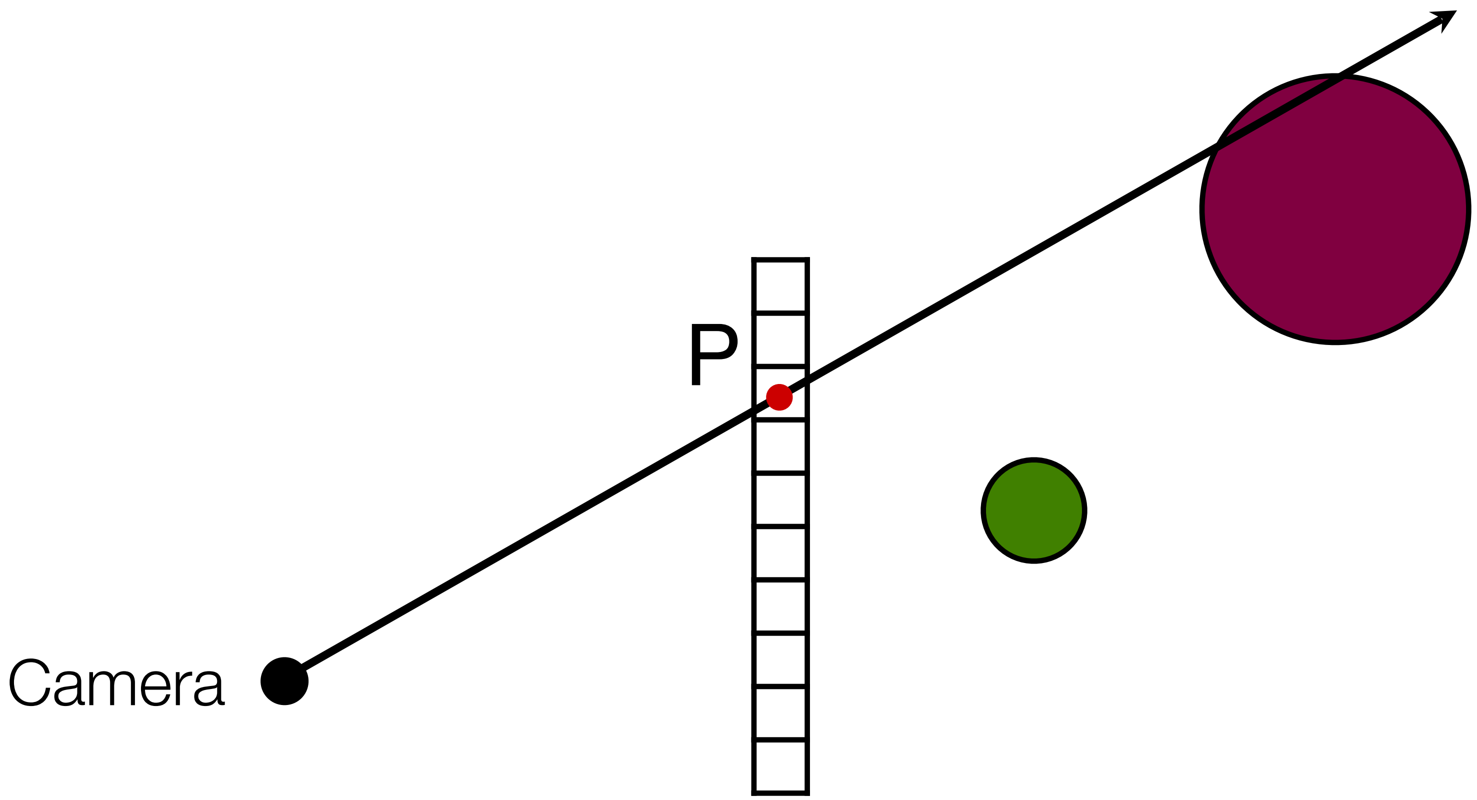
$$F(P) \bigcup G(P) = \min(F(P), G(P))$$

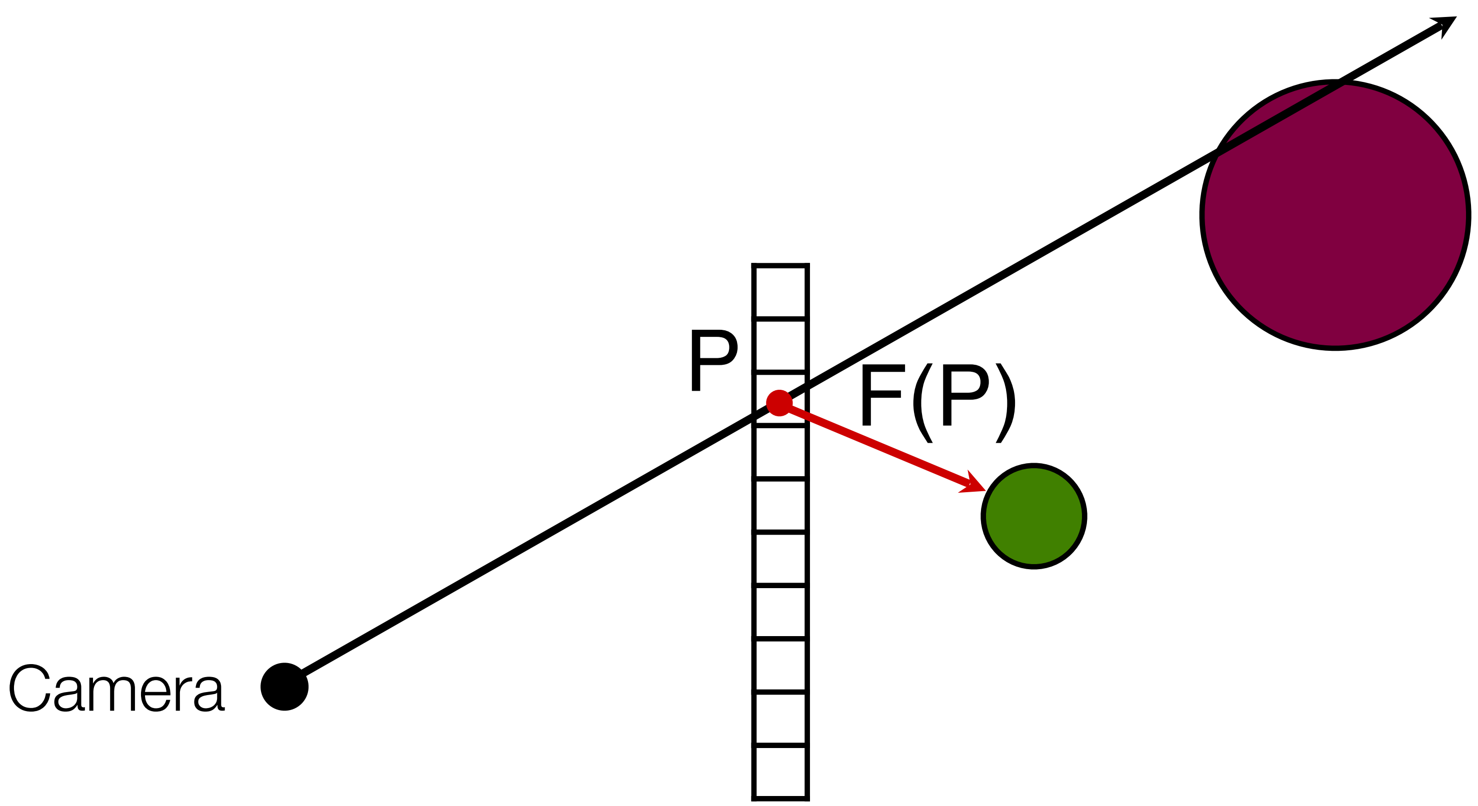
$$F(P) \bigcap G(P) = \max(F(P), G(P))$$

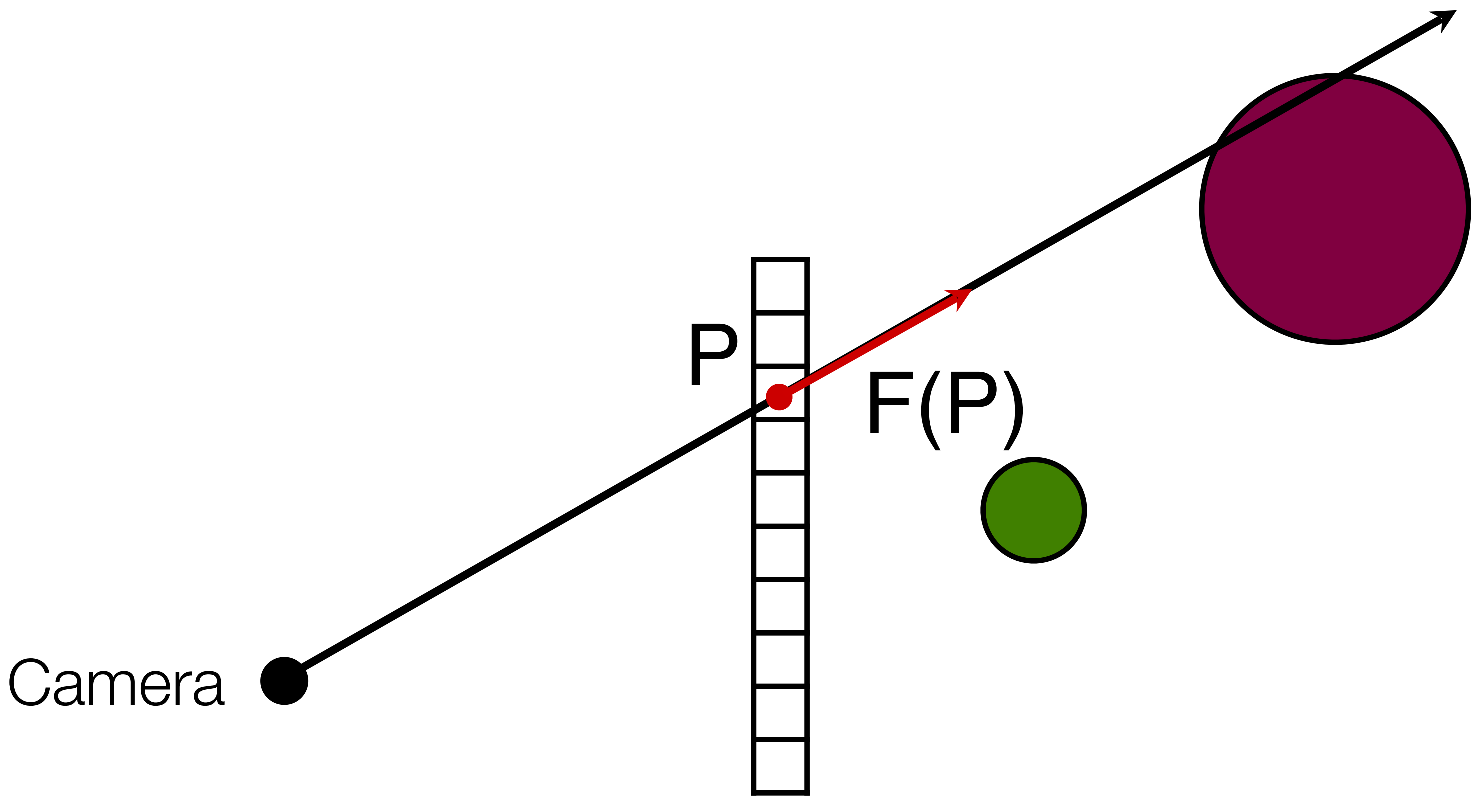
$$F(P) - G(P) = \max(F(P), -G(P))$$

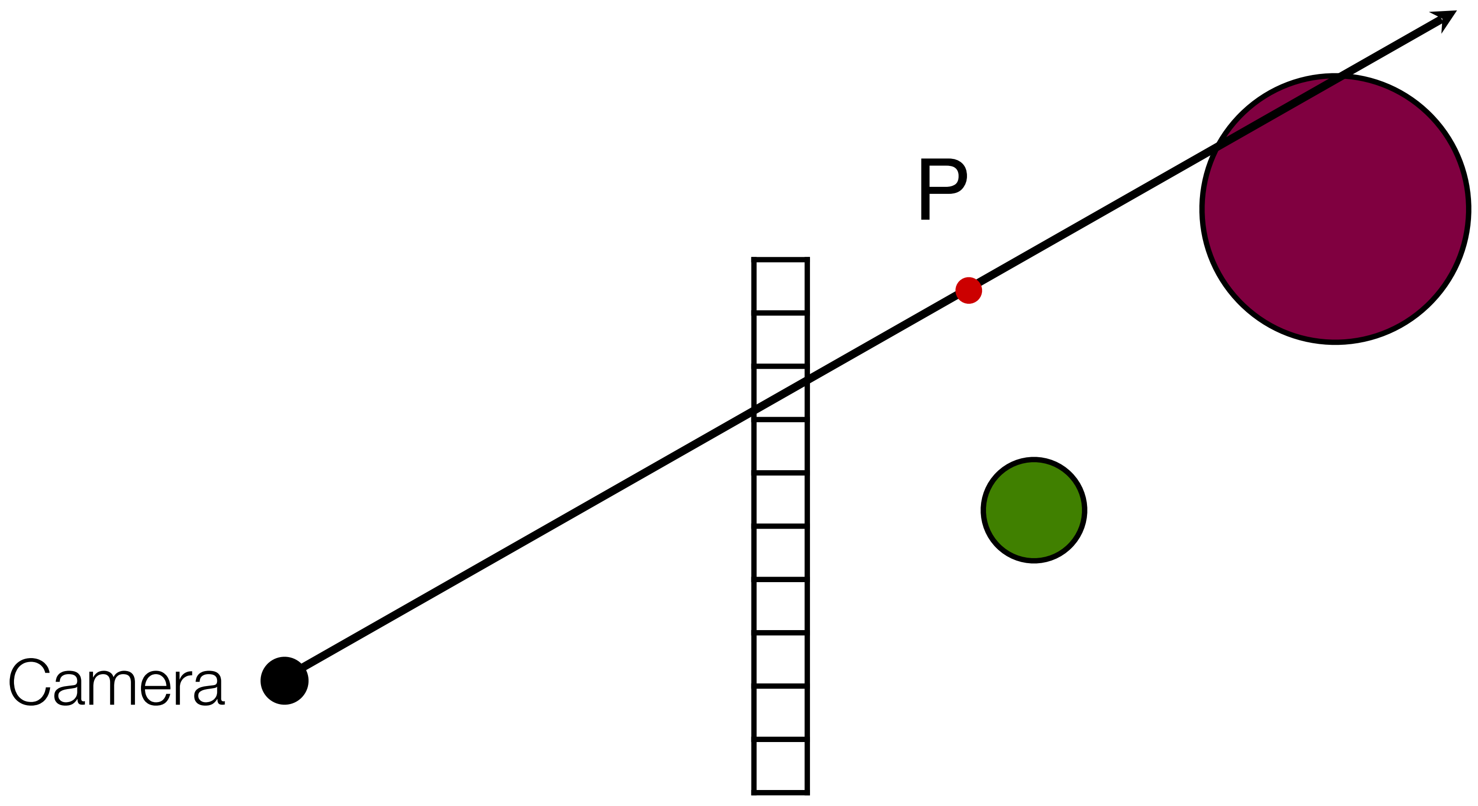
Signed distance functions

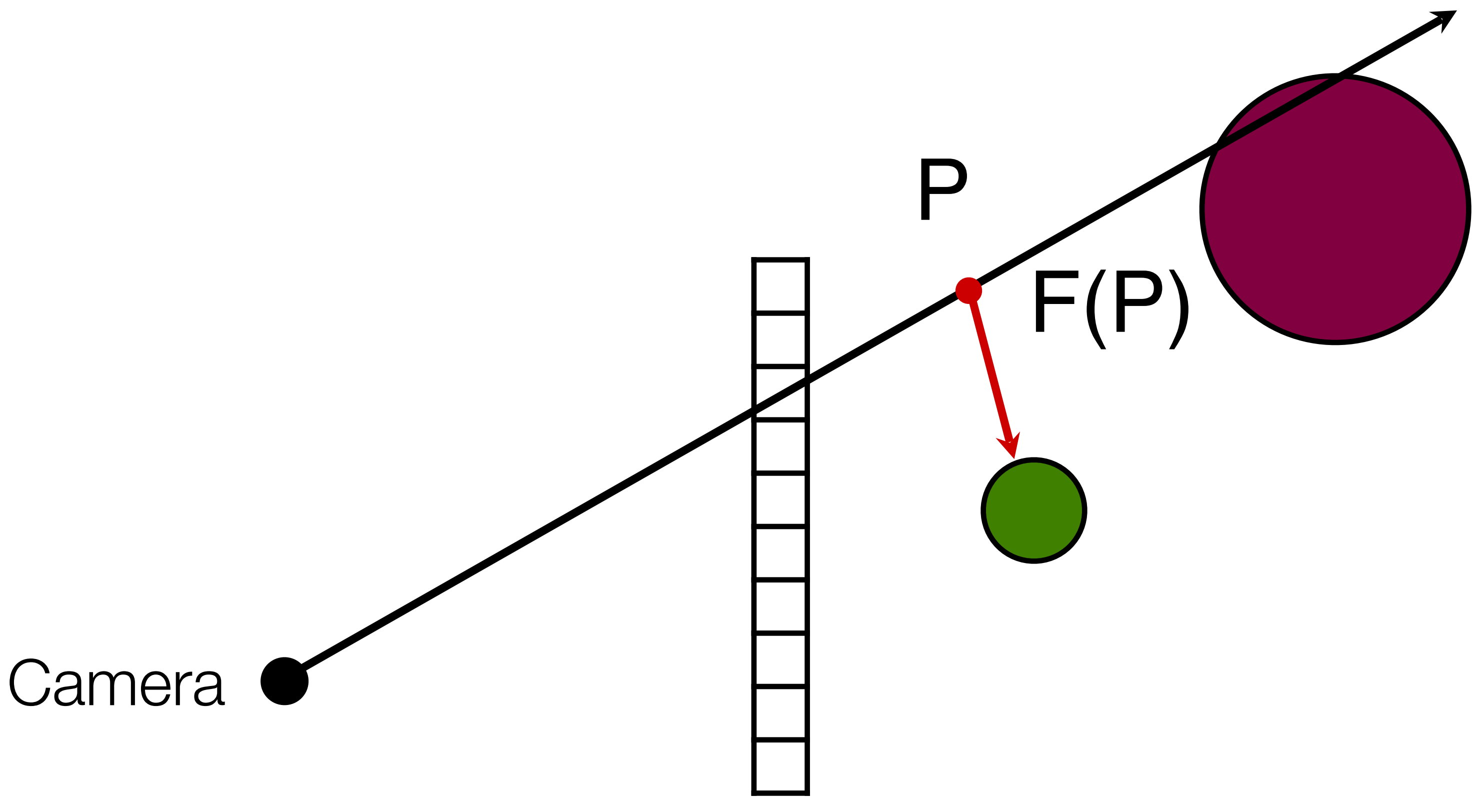
- We can render with these signed distance functions using a form of volumetric ray tracing called raymarching.
- In raymarching we treat the entire scene as one signed distance function.

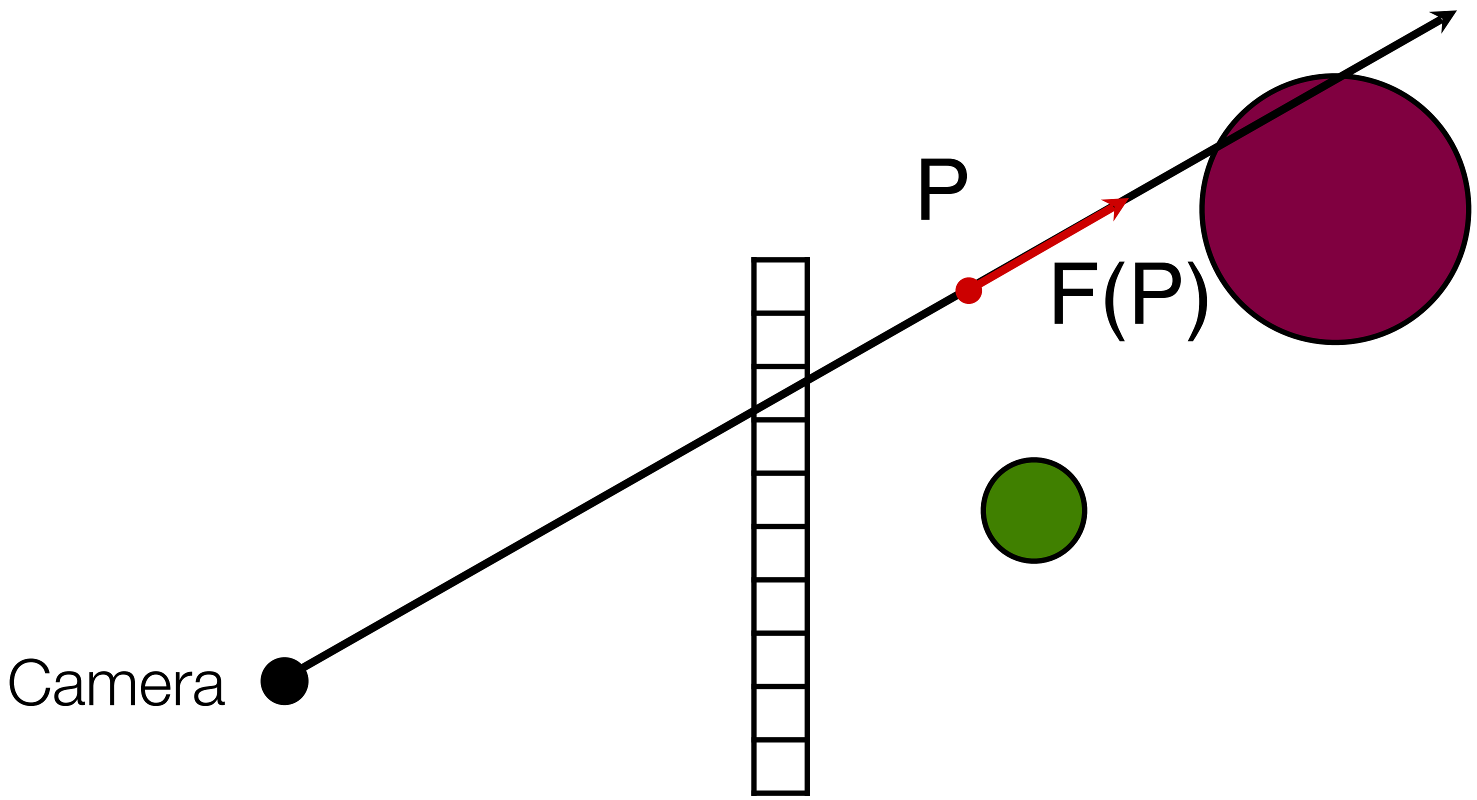

P
Camera

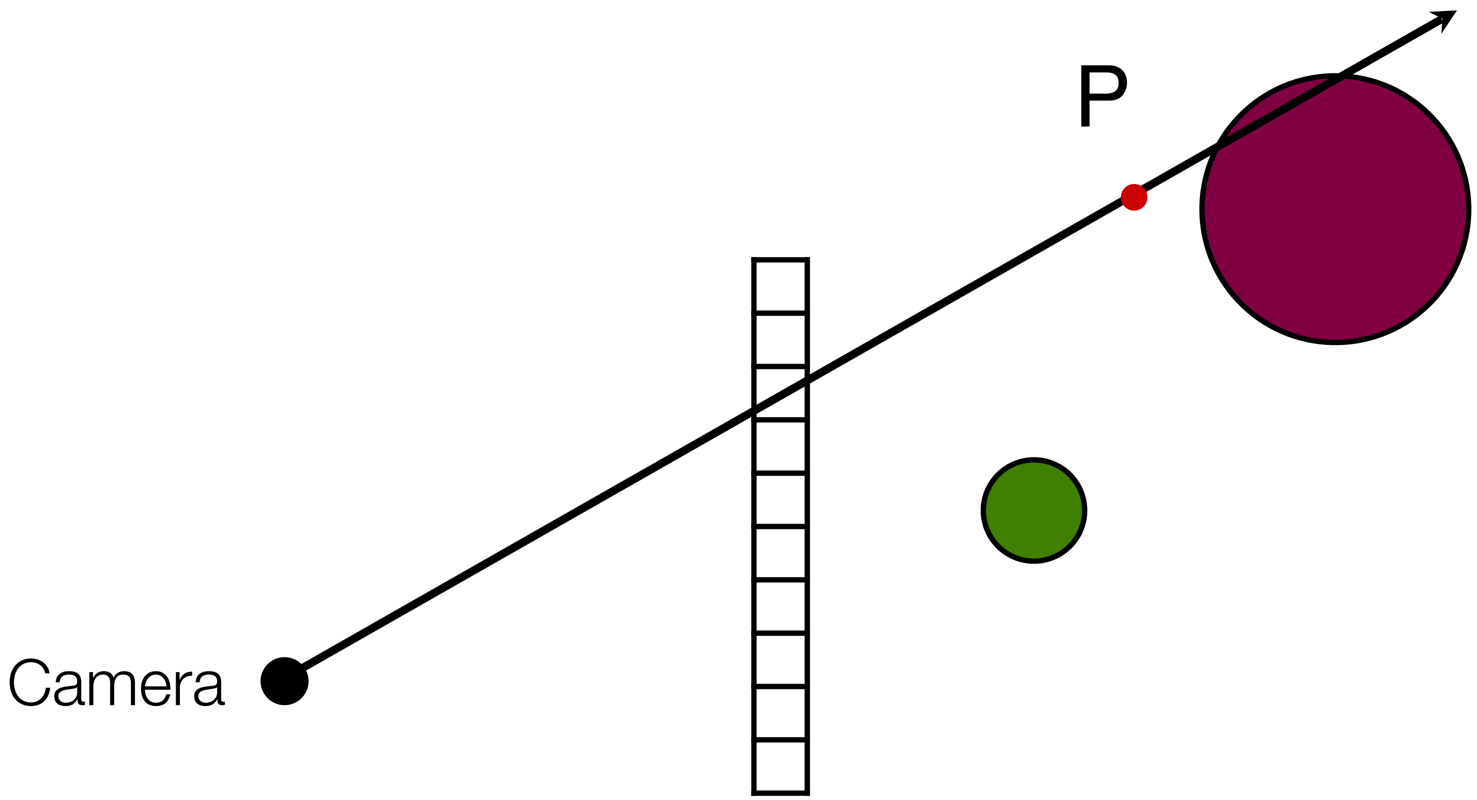

P
F(P)
Camera

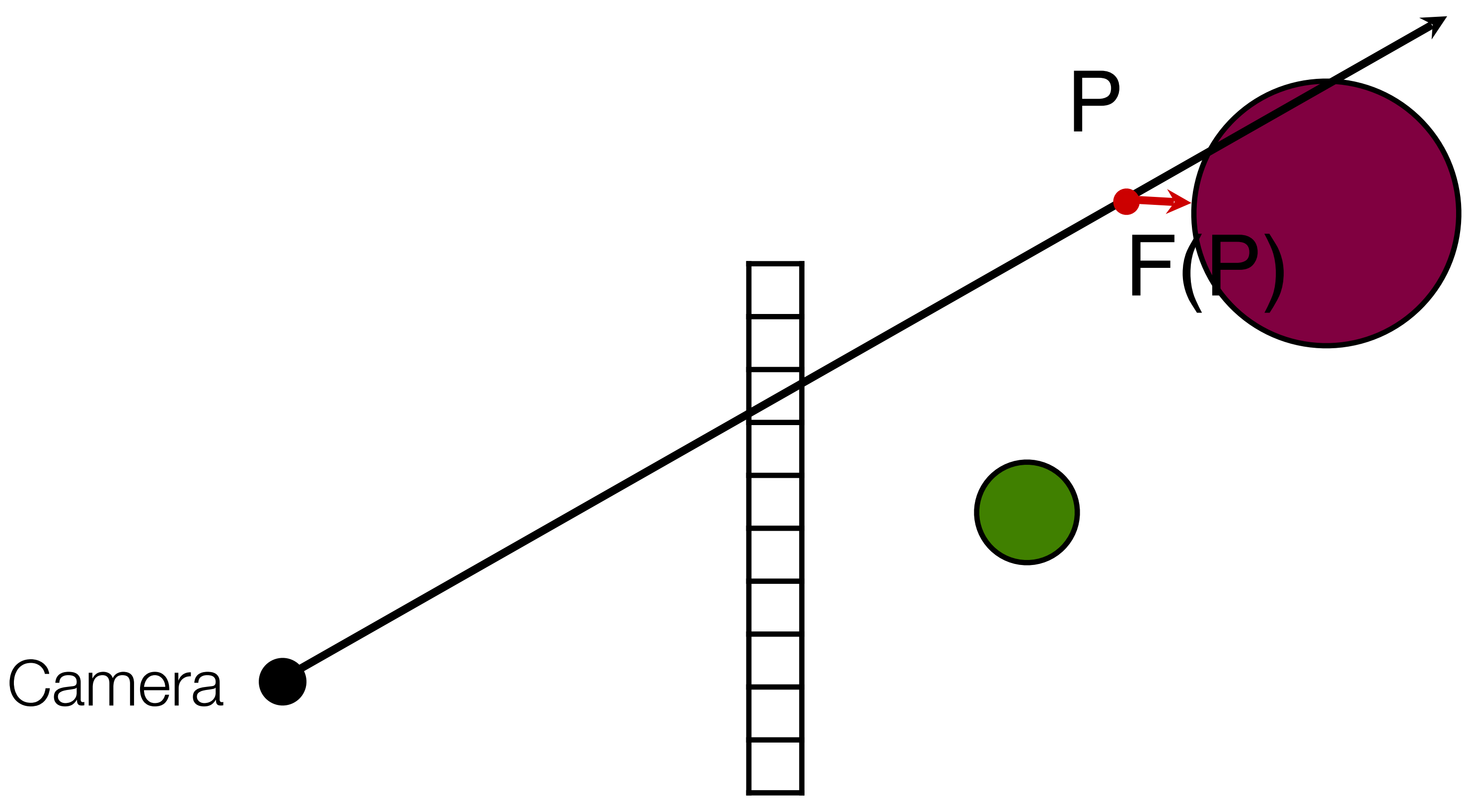

P
F(P)
Camera


P
Camera

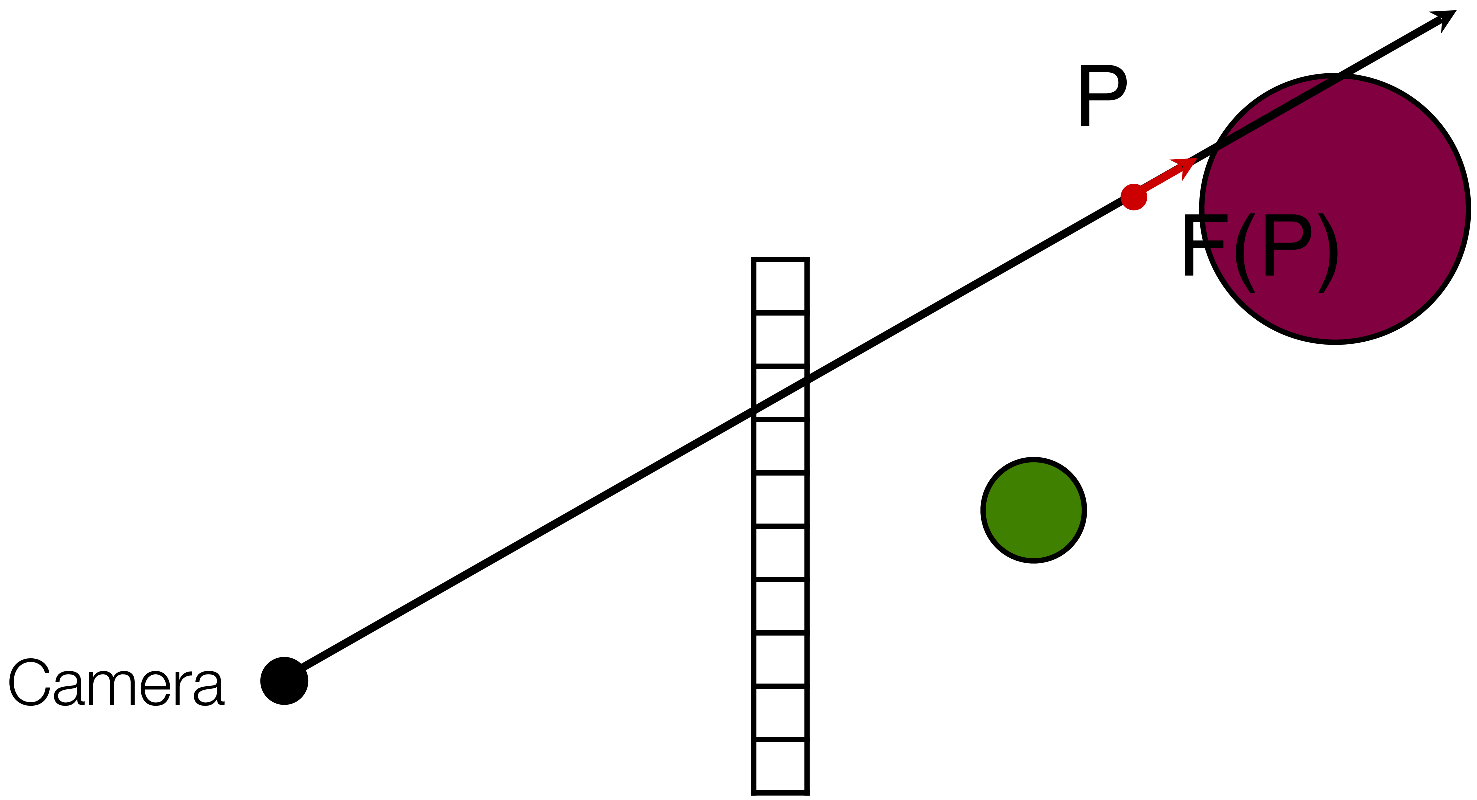

P
F(P)
Camera

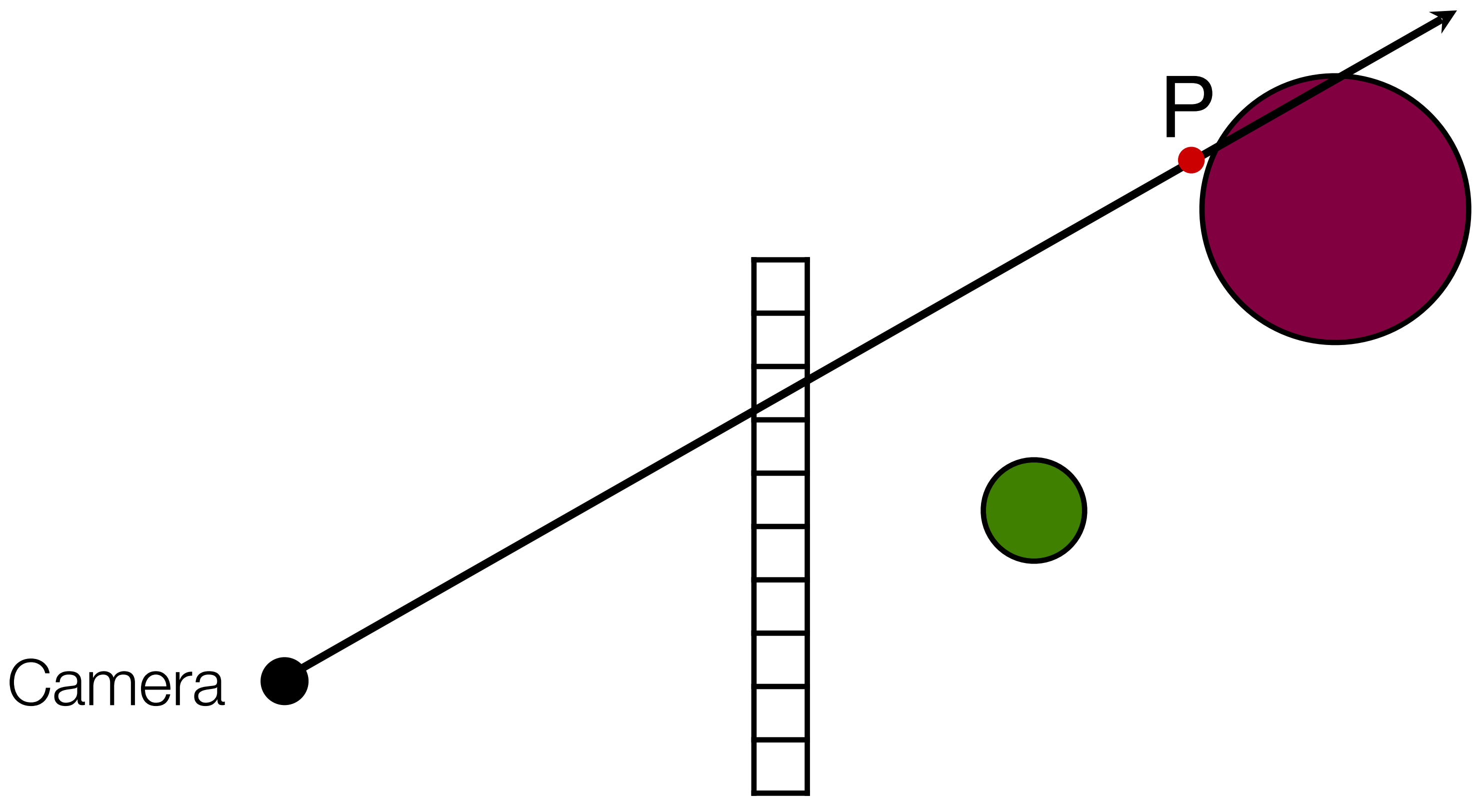

P
F(P)
Camera


P
Camera



P
F(P)
Camera


P
F(P)
Camera


P
Camera

Signed distance functions

- This technique gives us a way to do ray tracing without calculating the intersection of rays with the objects in the scene. It also has many other nice properties.

Signed distance functions

- Further reading:
- http://jamie-wong.com/2016/07/15/ray-marching-signed-distance-functions/
- http://iquilezles.org/www/articles/distfunctions/distfunctions.htm
- http://www.opencsg.org/

Signed distance functions

- Examples:
- https://www.shadertoy.com/view/Xds3zN
- https://www.shadertoy.com/view/4sS3zG
- https://www.shadertoy.com/view/MdBGzG
- https://www.shadertoy.com/view/MdX3zr
- https://www.shadertoy.com/view/3lsSzf

Voxels

- Basically, pixels in 3D.

Voxels

- Space problem:

$$
\begin{aligned}
&1000 \times 1000 \times 1000 \text{ voxels} \\
&= 1{,}000{,}000{,}000 \text{ voxels} \\
&= 1{,}000{,}000{,}000 \times 4 \text{ floats} \\
&= 4{,}000{,}000{,}000 \text{ floats} \\
&= 4{,}000{,}000{,}000 \times 4 \text{ bytes} \\
&= 16{,}000{,}000{,}000 \text{ bytes} \\
&\approx 16 \text{ GB}
\end{aligned}
$$

Voxels

- Because of this space requirement, naive voxel based rendering is impractical, even for non-realtime situations.
- In practice, voxels need to be stored in data structures which can represent large areas of empty space in an efficient way.

Sparse data structure

- A data structure that only represents values that are non-empty is called a sparse data structure.

- For voxels, one solution would be to use a sparse array.

- {((0,0,0),(1,1,1)),((1,4,5),(1,0,0)),....}

Voxel at (0,0,0) is white

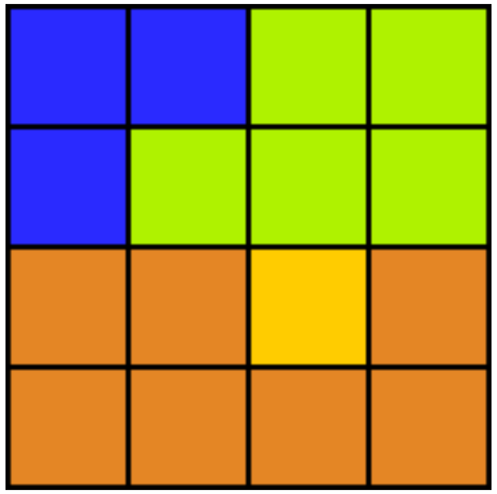
Voxel at (1,4,5) is red

Sparse data structure

- A sparse array is not suitable for rendering. For that we need a more complex data structure. We'll start by examining the problem in 2D then lift it to 3D.

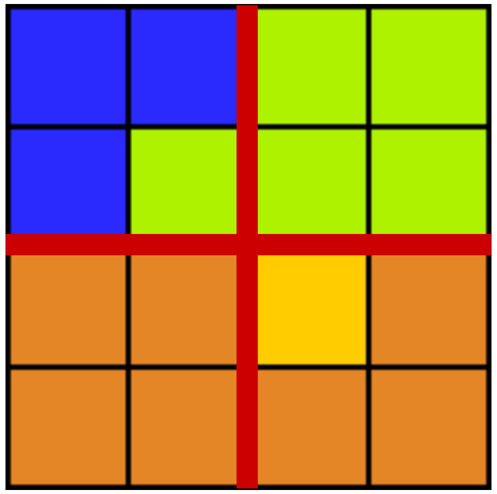
Sparse data structure

- This 4x4 image has 16 pixels, but there are large patches where the pixels are the same colour and we want to take advantage of that.



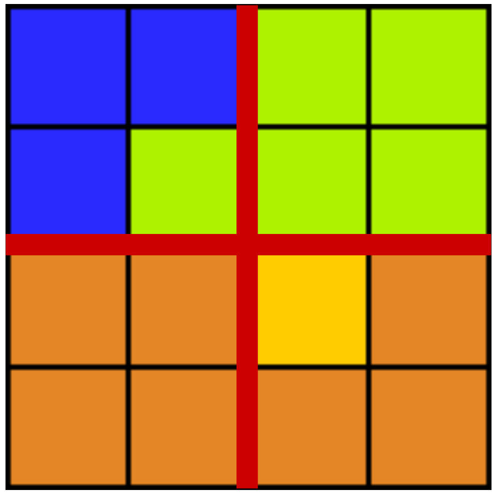
Sparse data structure

- We start by dividing the image up into 4 sections.



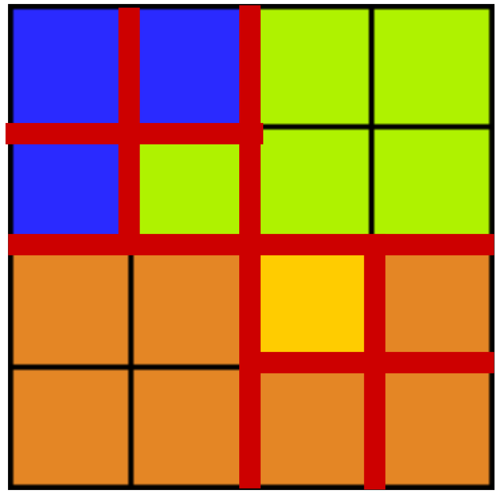
Sparse data structure

- Of the four sections, two are uniform in colour. We leave those two sections alone. The other sections we recursively divide into 4 and repeat the process.



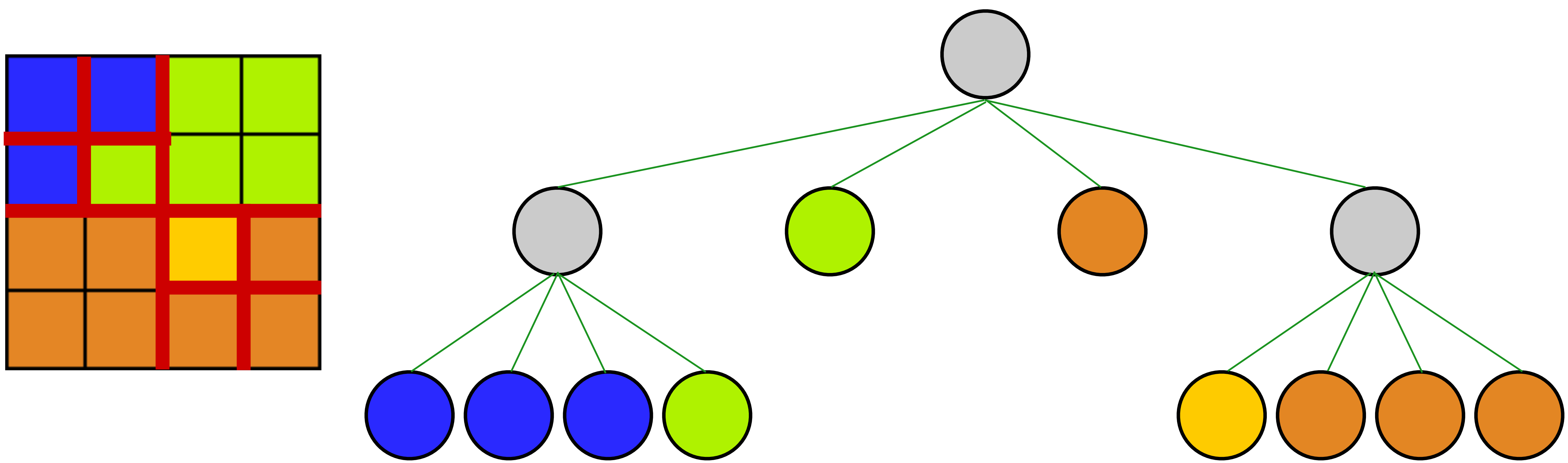
Sparse data structure

- We continue this process until we reach the individual pixel level.



Quadtrees

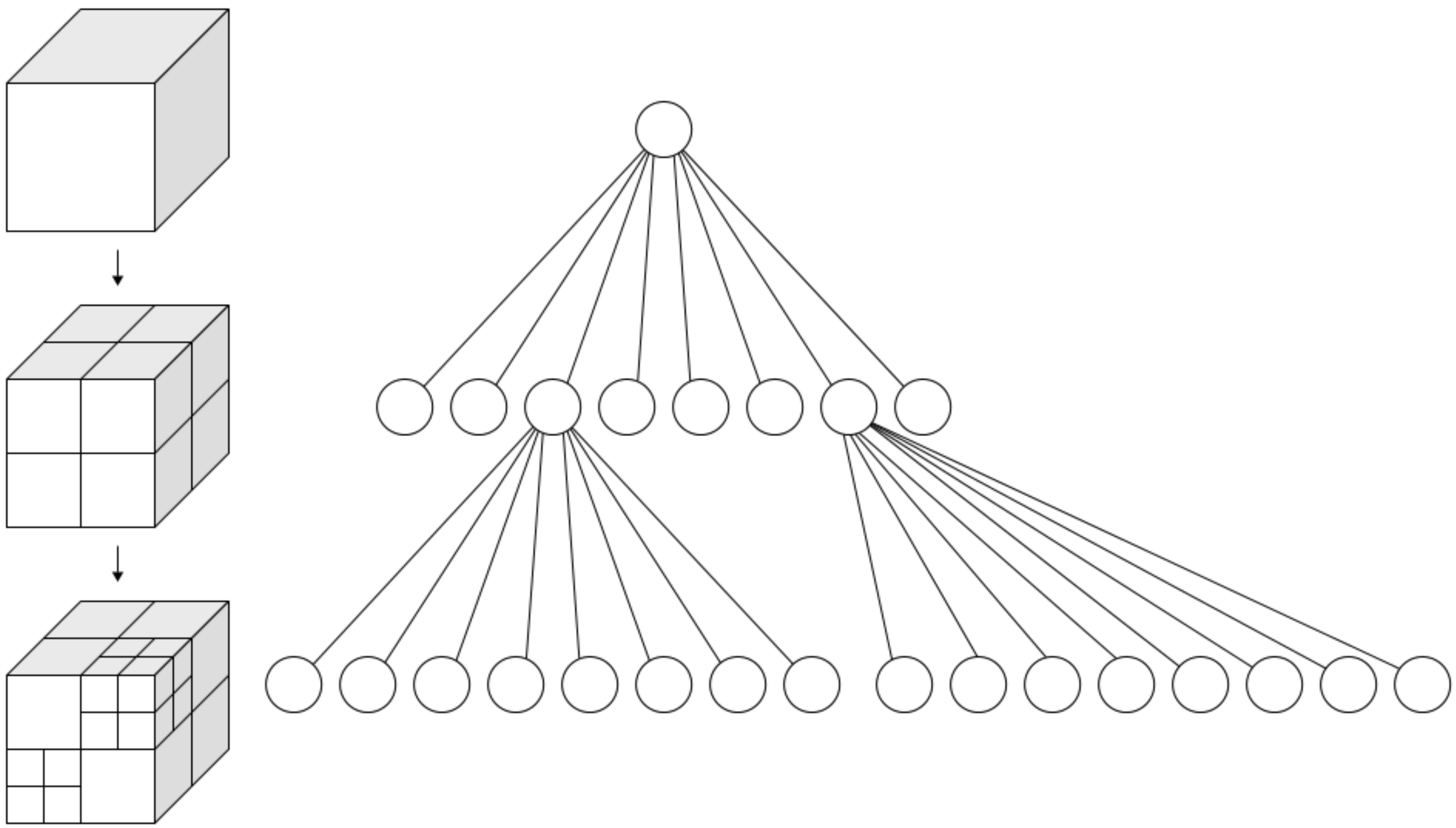
- We can represent this as a tree.



- This is a quadtree.

Octrees

- Generalising this to 3 dimensions gives an octree.



Octrees

- Rather than store a single voxel at the leaves of the tree, we typically store a block. This gives the best of best of both worlds, the space efficiency of a sparse data structure with the cache-friendly memory access of a contiguous block of data.

Voxel

- Further reading:
- https://medium.com/retronator-magazine/pixels-and-voxels-the-long-answer-5889ecc18190
- https://www.nvidia.com/object/nvidia_research_pub_018.html

Voxels

- Examples:
  - https://www.shadertoy.com/view/4dfGzs
  - https://www.shadertoy.com/view/XtXyDS
  - https://www.youtube.com/watch?v=TjmRPjnWJ5g
  - http://qake.se/demo/

VR & AR

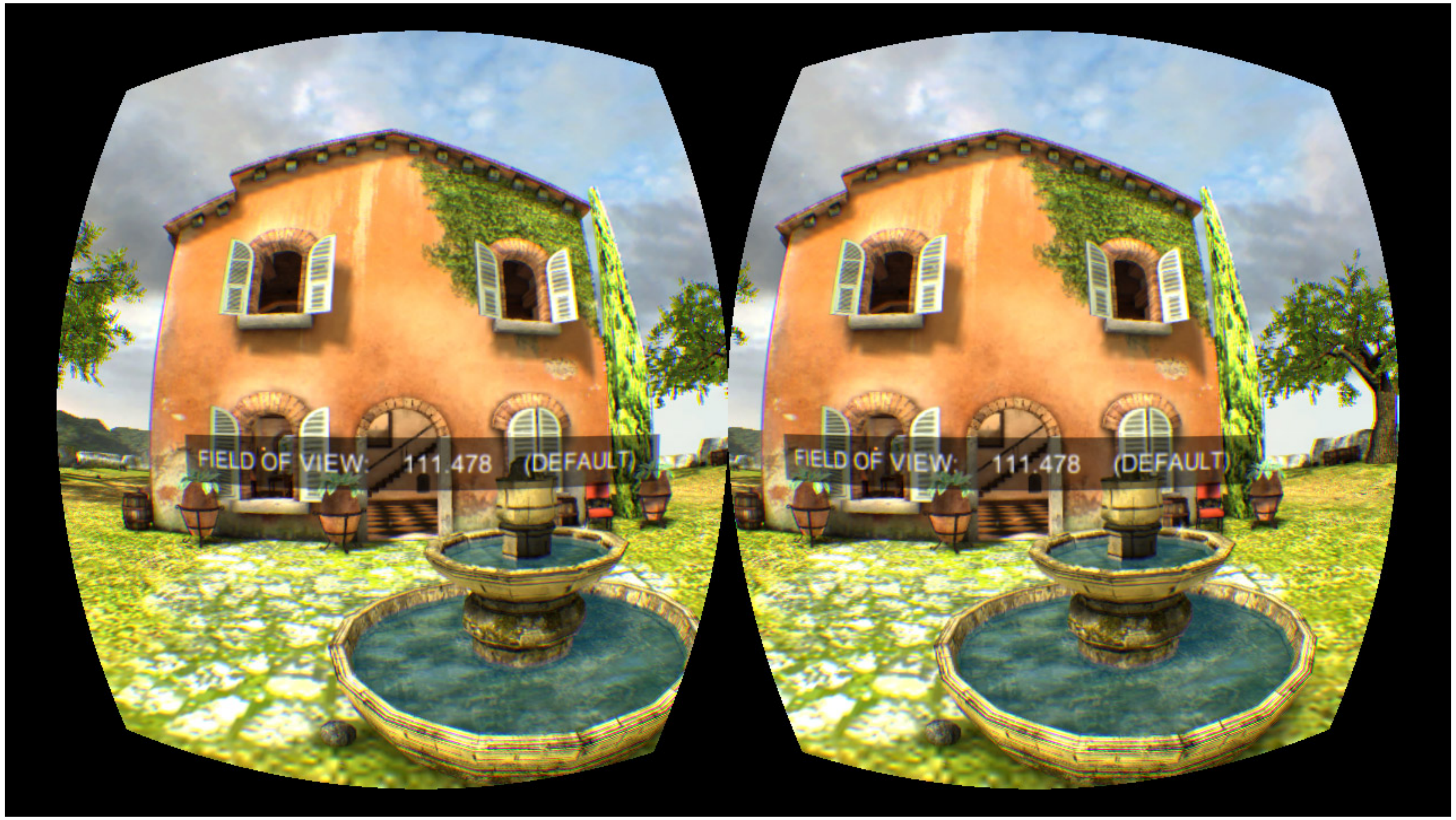
- Virtual Reality
    - A completely simulated virtual world intended to replace the users vision.
- Augmented Reality
    - Enhancing the real world with virtual features.

Rendering Scenes for VR

- Need to render twice. Once for each eye.
- The average length between human eyes is 65mm.
- Render for the one eye then translate the camera to render for the other eye.

Rendering a Scene for VR

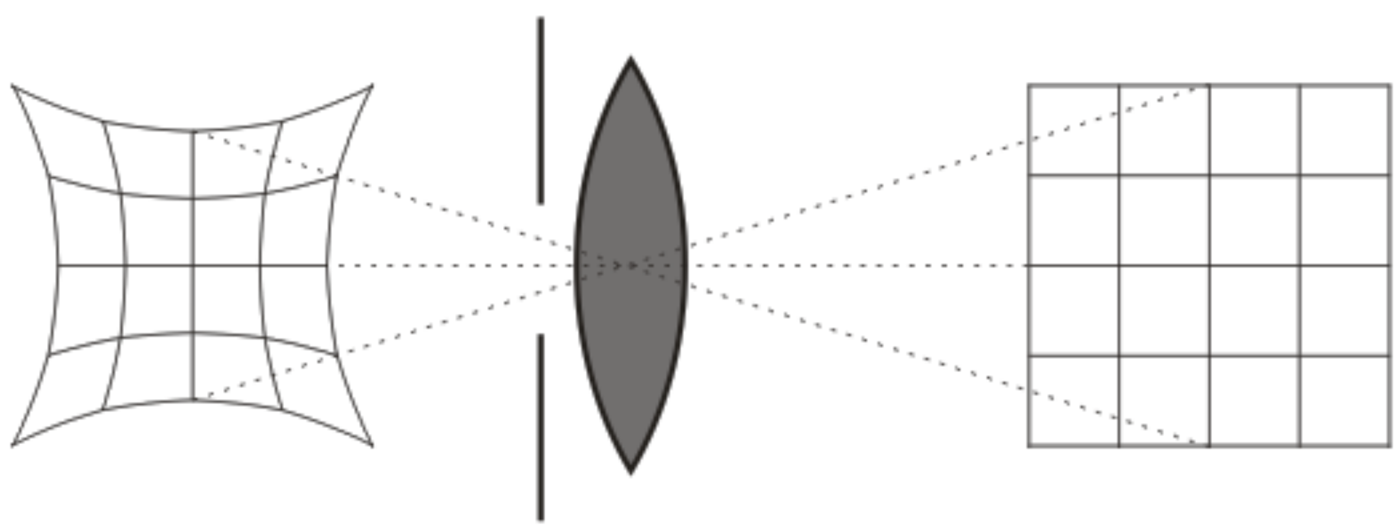
- Need to have a projection with a wide field of view and to also account for lens distortion



Lens Distortion

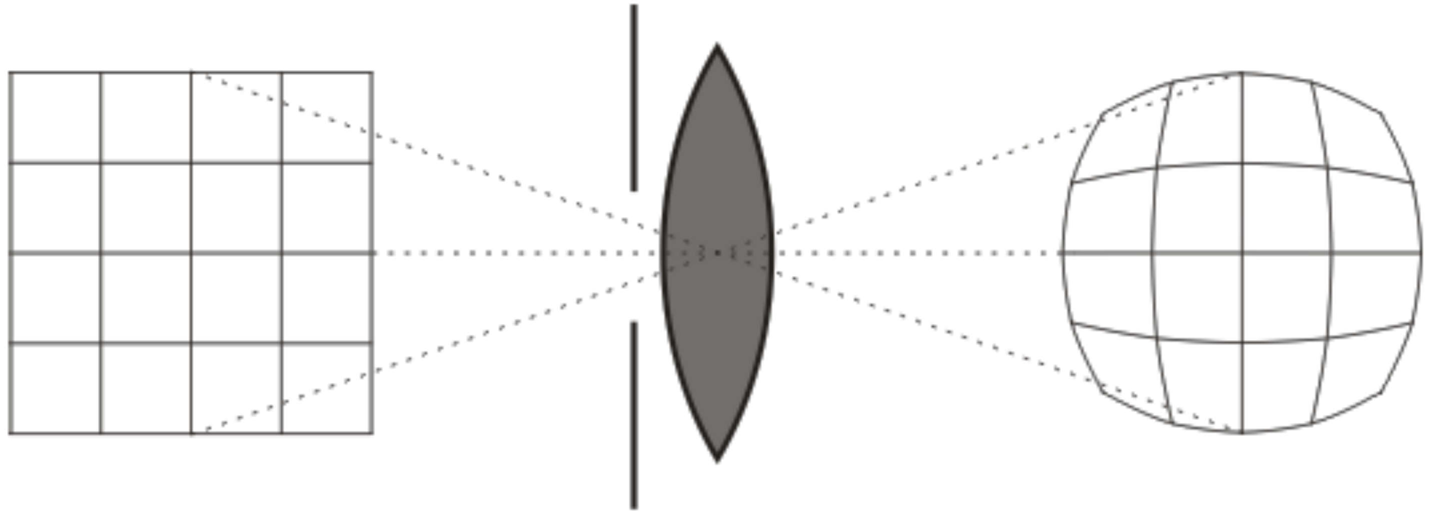
- VR headsets contain lenses that are designed to show as much of the screen in the person's field of view as possible.

- These lenses also distort what is on the screen.



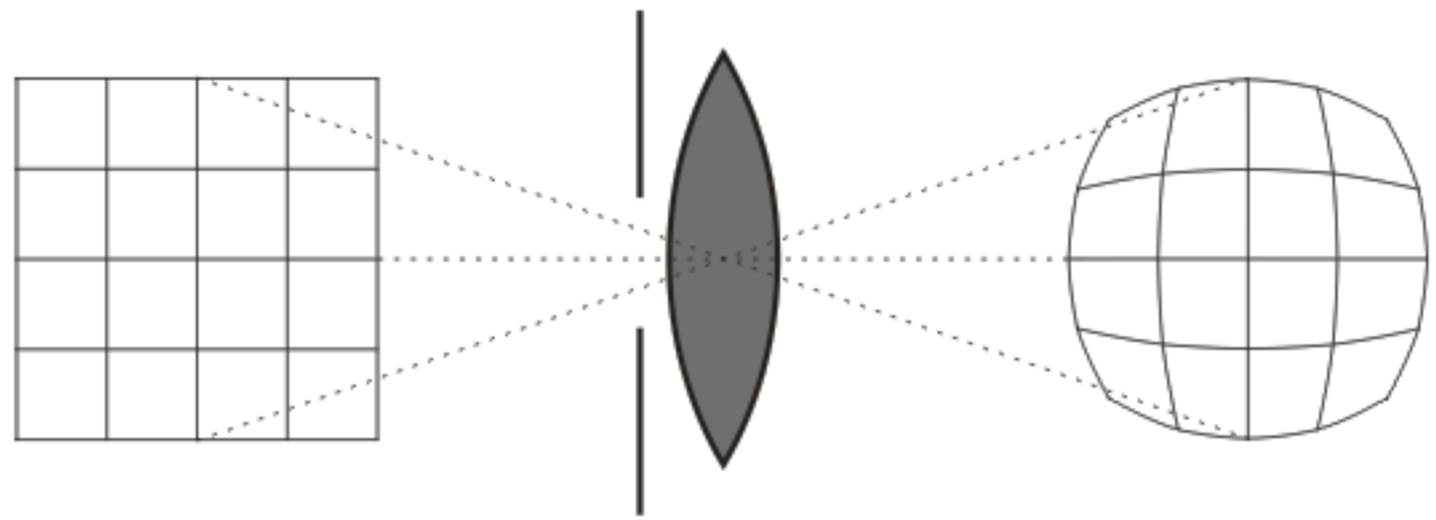
Lens Distortion

- Developers need to counteract that by drawing an image to the screen with reverse-lens distortion.



Lens Distortion

- This distortion is a non-linear transformation. It does not preserve straight lines.

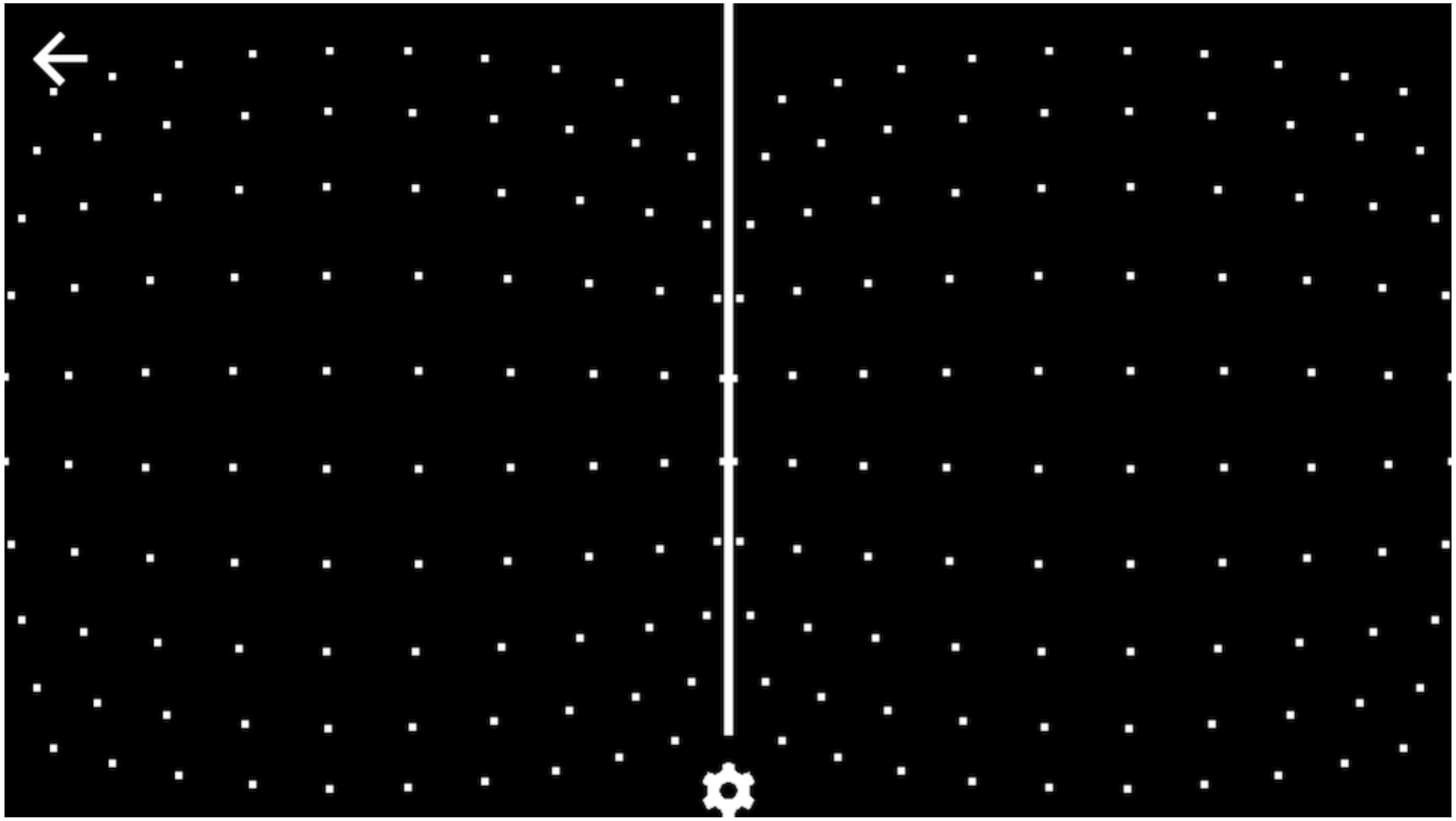


Lens Distortion

- One approach is to use postprocessing. Render the scene to a texture then apply a filter that does the distortion by moving pixels closer to the center.
- This is slow and causes pixels to be lost, reducing detail.

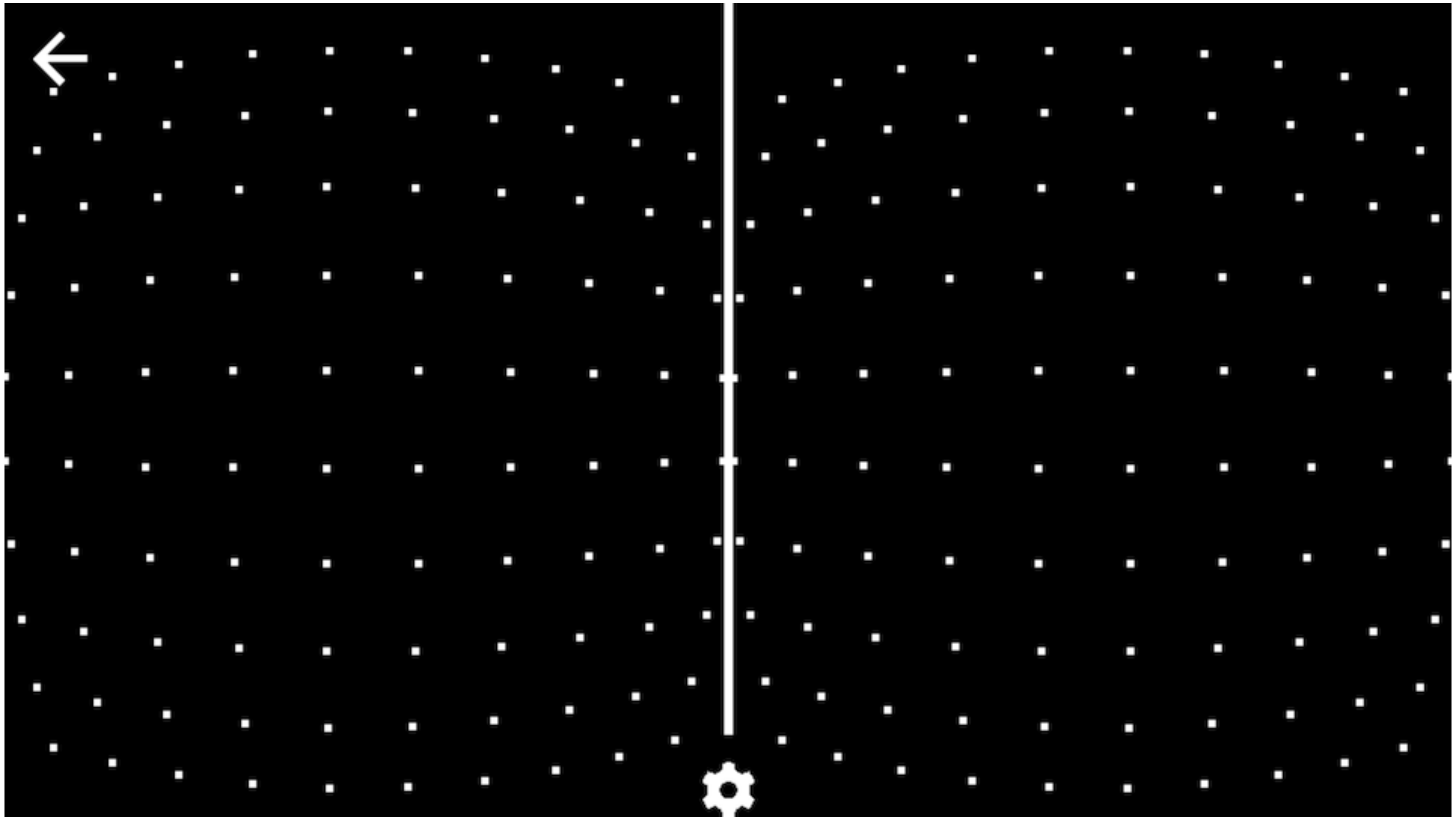
Lens Distortion

- Another approach is to render the scene then apply it as a texture to a low resolution barrel shaped mesh.



Lens Distortion

- This works, but still requires two rendering passes.



Lens Distortion

- The fastest approach is distort the vertices in a vertex shader and render the scene as normal.
- However, this doesn't distort straight edges.

Lens Distortion

- Large polygons need to be tessellated so that after distortion the result approximates a reverse-lens distortion.

Lens Distortion

- Because the tessellation results in higher vertex density, Gouraud shading often looks just as good as Phong shading. Google cardboard uses this approach to do VR on less powerful hardware.