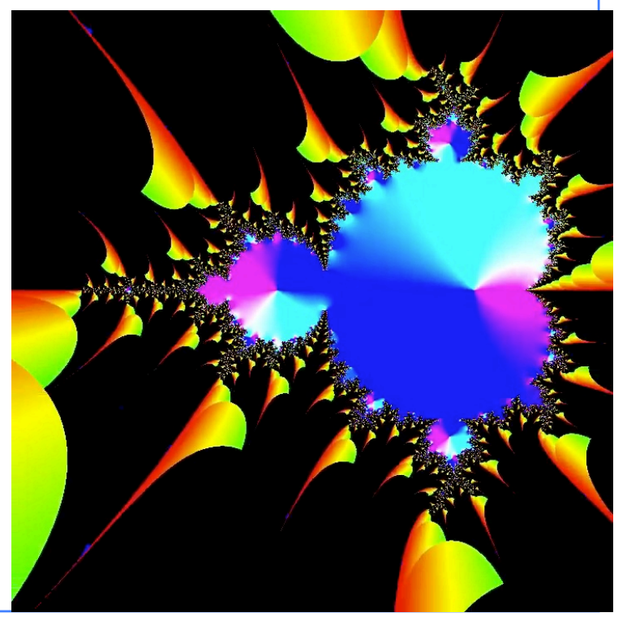
TSBK07/TSBK11

Computer Graphics

Ingemar Ragnemalm, ISY



Lecture 12

Collision detection and response

Fractals

Time to decide on a project

- Should be preliminary decided wednesday the 5th (second to last lecture)
- Final specification same day as last lecture (friday 7th)

Title
Participants
Brief summary
Will-do
Might-do

Project ideas:

(The more basic ones)

TSBK11: Look here: ->

- 3D labyrinth
- Interactive solar system
- Driving simulator
- Flight simulator
- Enhanced terrain rendering

Rather TSBK07: ->

- Robot (or Godzilla) with moveable limbs
- Large virtual reality with frustum culling/level of detail/portals/etc

Important: Anything course relevant that you feel is interesting is valid!

Unfinished labs?

Don't panic!

2 sessions left, but also:

The course does not end here.

"Project sessons" thursdays VT2.

Project work, but labs are also welcome.

Additional tools reminder

1) **SimpleGUI**: GUI controls in an OpenGL window

One source file, minimal dependencies

2) Code for **simpler upload of shader data**

uploadMat4ToShader() etc, in the VectorUtils files.

3) **Geometric Utilities (GLUGG)** for procedural geometry.

4) **LoadTextures**, a bigger texture loader (4 formats)

5) **SimpleFont** and **my-stbtt**, for drawing text

6) **Call me AL**, sound playing API using OpenAL

Lecture questions

1. How can spheres vs polyhedra be a really simple but working collision detection?

2. How do you calculate the collision response between a sphere and a flat stationary surface?

3. Suggest a global phase collision detection algorithm

4. What the heck does 2.5 dimensions mean?

5. How can I create a procedural tree?

Collision detection

Broad phase - narrow phase

Enclosing shapes

SAT

Containment/intersection

Exact and approximate

Exact:

Broad/narrow

SAT for broad phase with OBBs

Containment/intersection, simple but not fast narrow method

Moore&Williams, 1988

Approximate:

Simplified narrow phase, or no narrow phase

Hierarchies of spheres = sphere trees

Hubbard, Time-critical collision, 1996

That sounded hard!

Does it have to be so complicated?

so let's talk about

Simplified collision detection

Sometimes it doesn't have to be hard at all!

Simplified collision detection

Simplified shapes

Sphere-polyhedra tests

AABB worlds

Approximative methods
Collision shapes

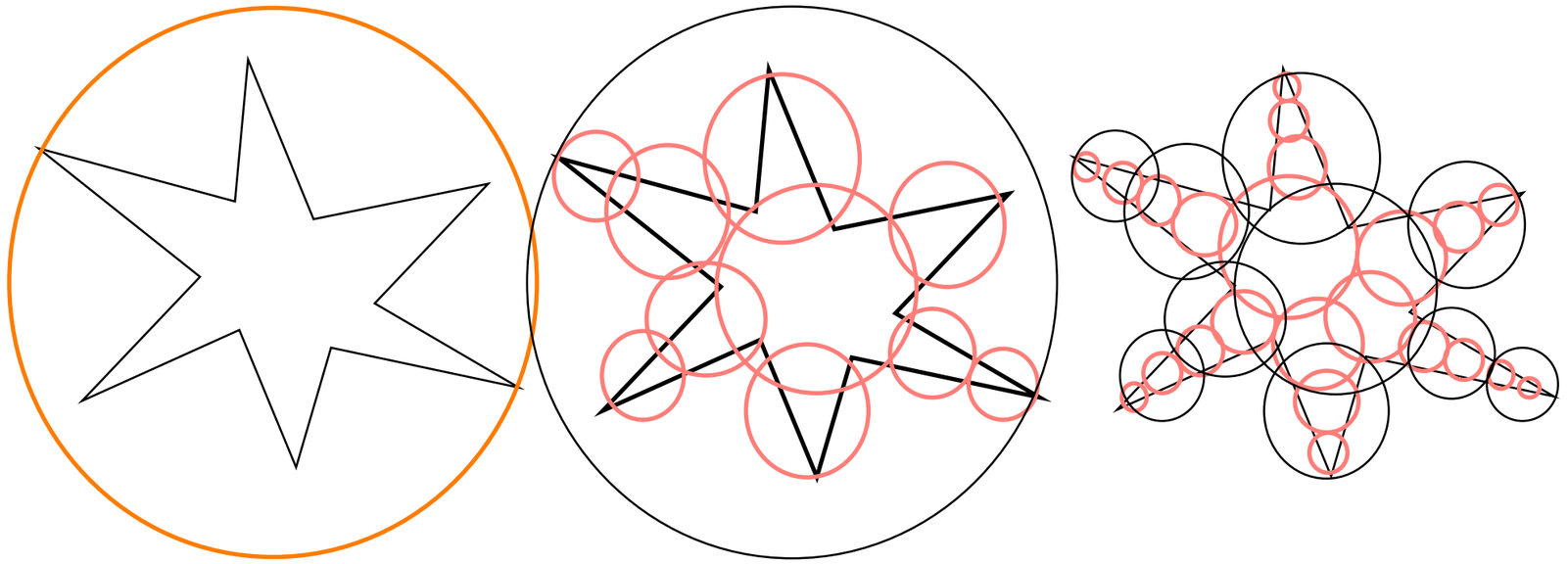
Simplify the narrow phase with simplified versions of the geometry

- Low-polygon version of a polyhedra
- A "standard" shape that follows the shape as closely as possible

The "narrow phase" more or less blends into the "broad phase"

Object hierarchies

"Sphere trees"



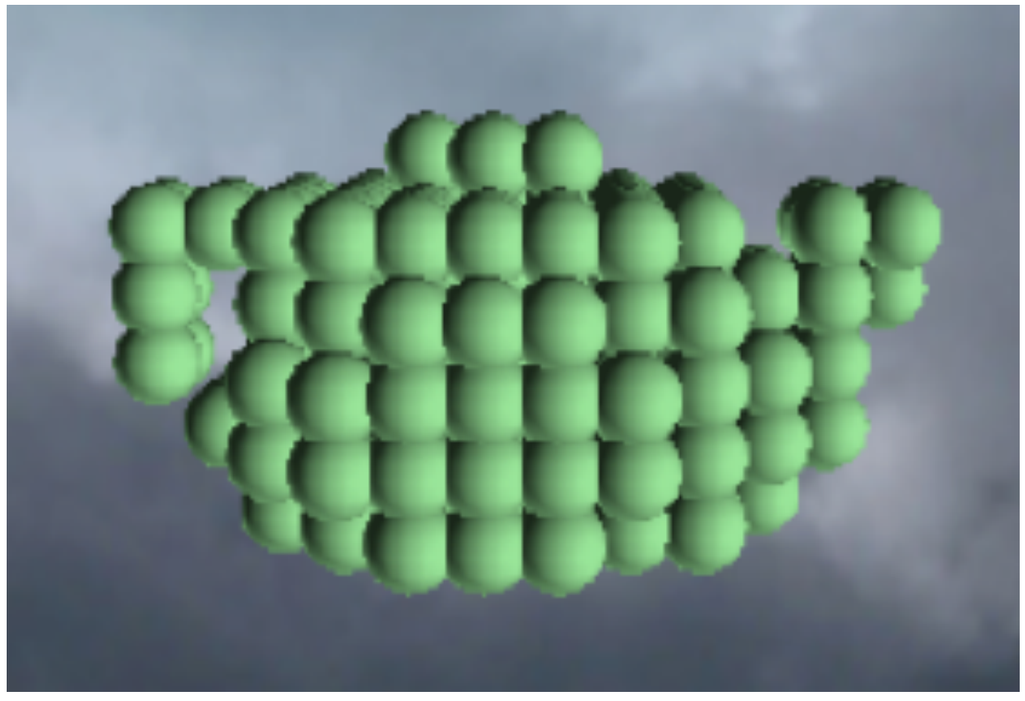
Collision detection is done to the level that available time permits

The hierachy can be produced using distance maps

Hubbard, Time-critical collision, 1996

Voxelization

Break down model to voxels, test with one sphere per one voxel

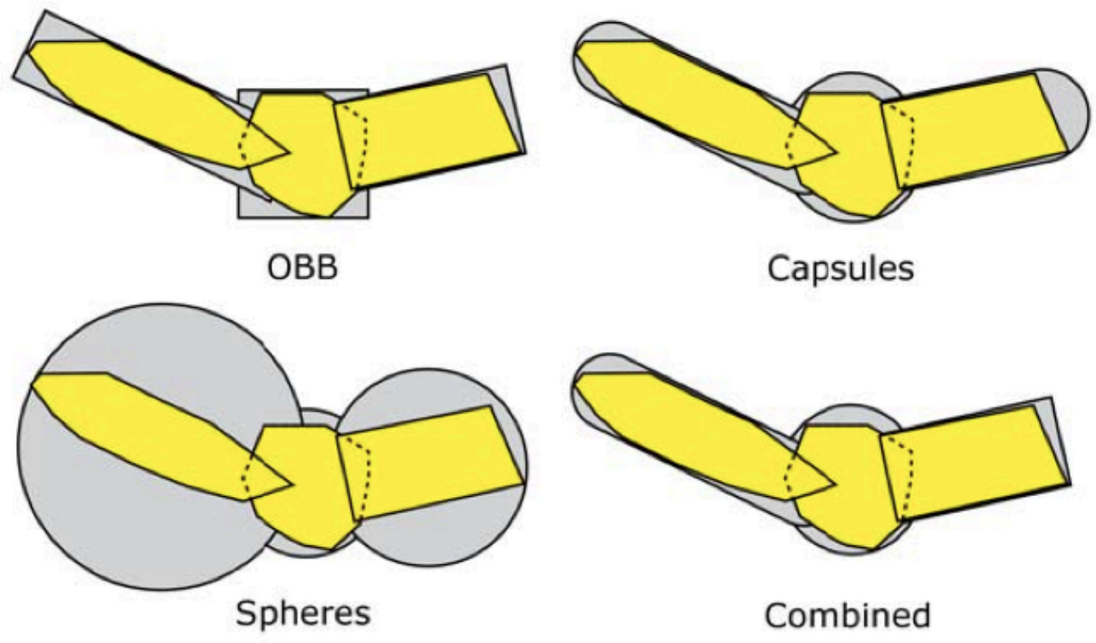


Teapot voxelized to 171 spheres

From Edhammer, "Rigid Body Physics for Synthetic Data Generation", 2016

Multiple simple shapes

Use a number of very simple primitive shapes to approximate a complex shape



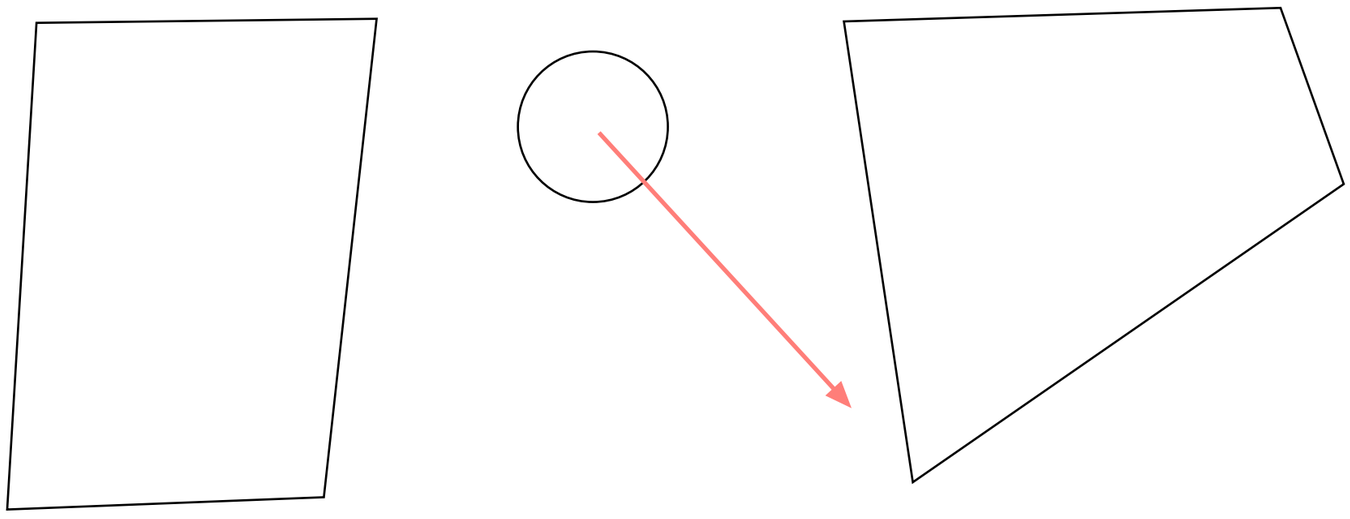
From:
Henrik Bäcklund,
Niklas Neijman,
"AUTOMATIC MESH DECOMPOSITION FOR REAL-TIME COLLISION DETECTION",
2013

HACD

Hierarchical Approximate Convex Decomposition

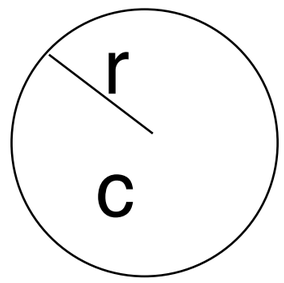
IMAGE REMOVED

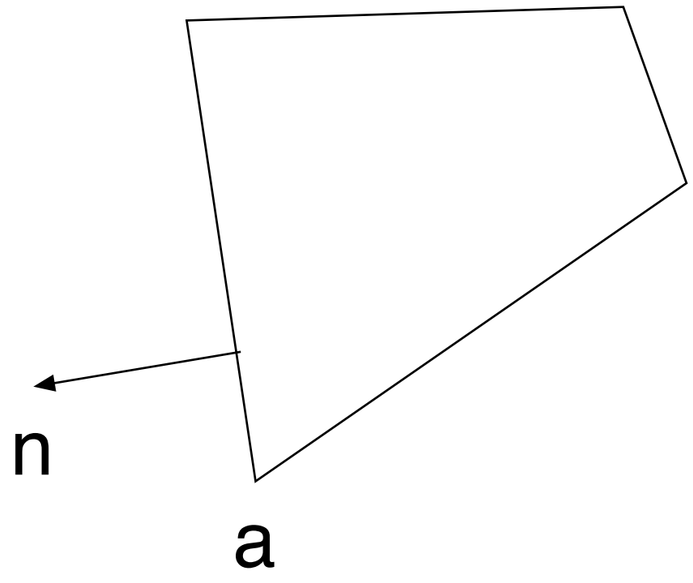
Spheres in polyhedra world: cameras as well as objects



The size gives us a minimum distance to walls!
(E.g. more than the near Z clipping distance.)

Sphere - polyhedra distance



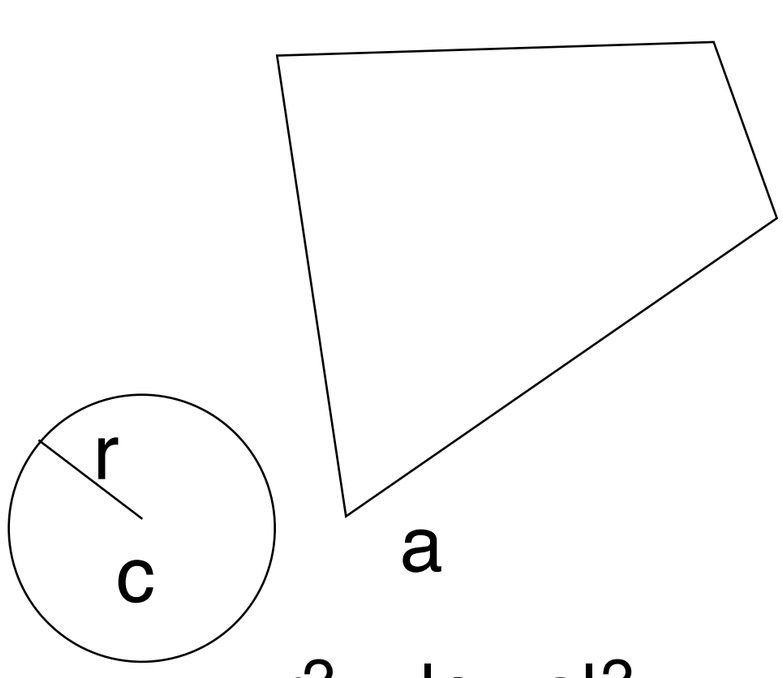


$n \bullet (c - r \cdot n) > n \bullet a$

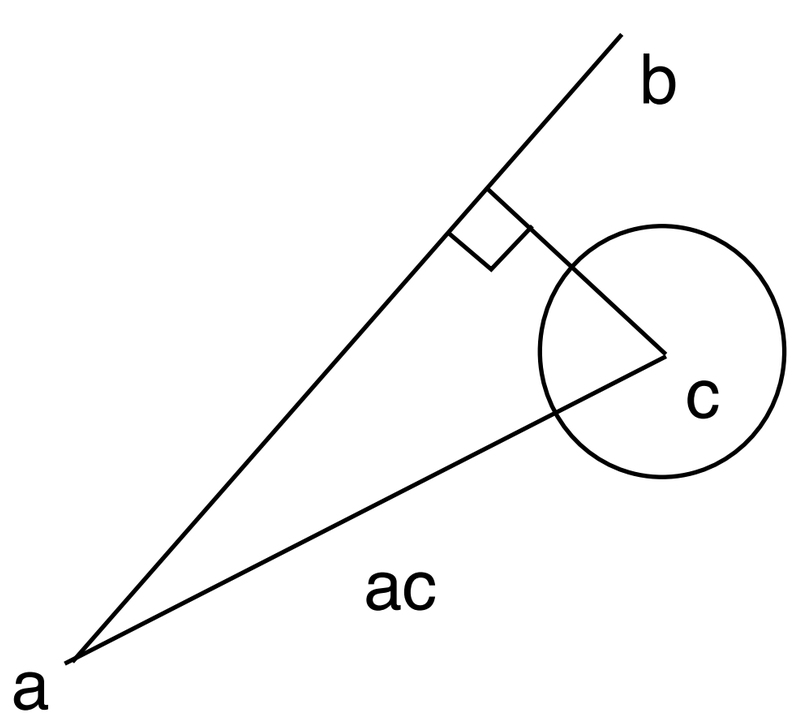
$\Rightarrow$ Distance from center to plane > r

$r < n \bullet (a - c)$ Valid when a line through c along n intersects the polygon!

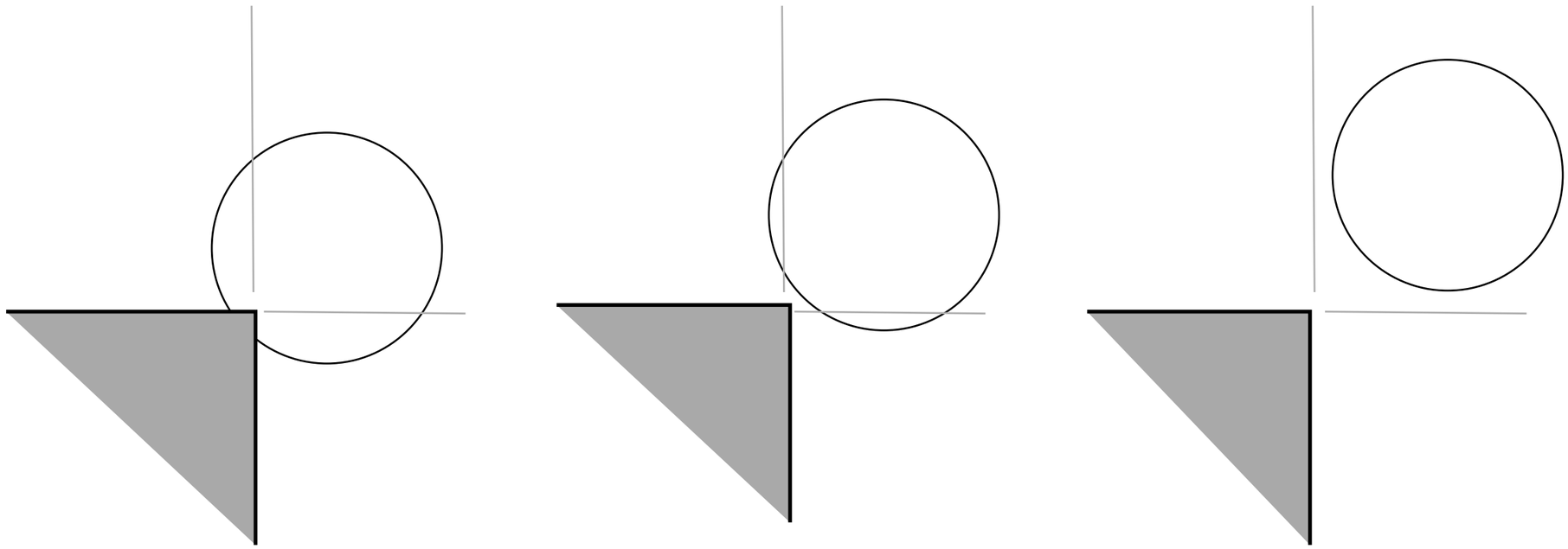
Exact test: Check corners and edges too



$r^2 > |c - a|^2$



Simplification: Test all planes only! If outside any plane - outside, otherwise it intersects.



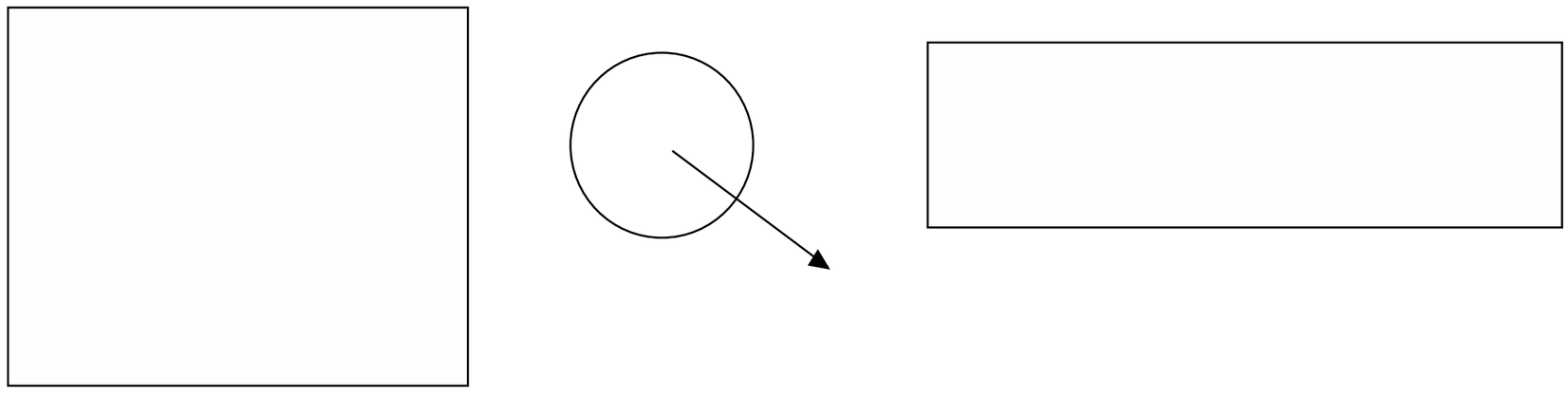
Correct hit

False hit

Correct miss

Simple case: Axis aligned 3D maze

All walls are AABBs All objects are spheres/cylinders



Final notes on the simplified camera/sphere-polyhedra collisions:

Resolving: Pick the closest intersected plane as the one to ”hit”, and use that for collision handing. The smallest change is usually correct.

Conclusion: Don’t overdo it if you can fake it. Be ambitious, but don’t waste time on effects that nobody will notice.

Global phase collision detection

Avoid many-to-many checks

Space subdivision

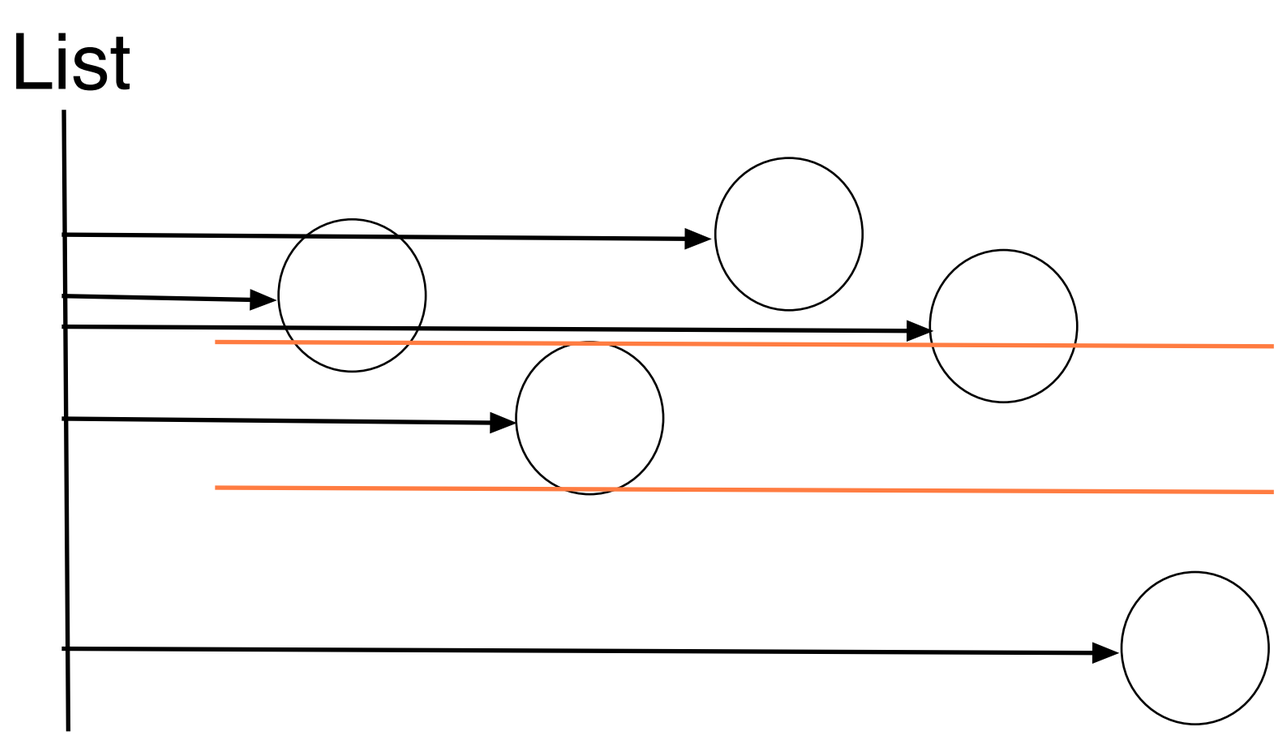
Simple: Linear sorting

More elaborate: Hierarchies, octrees…

Simplify objects out of view?

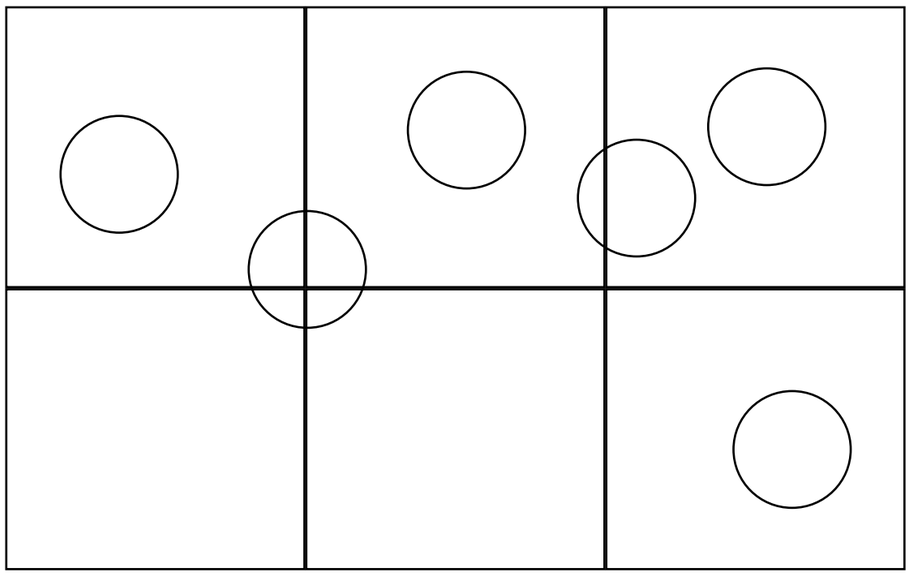
Linear sorting

Reduces the problem by 1 dimension



Subdivision for collision detection

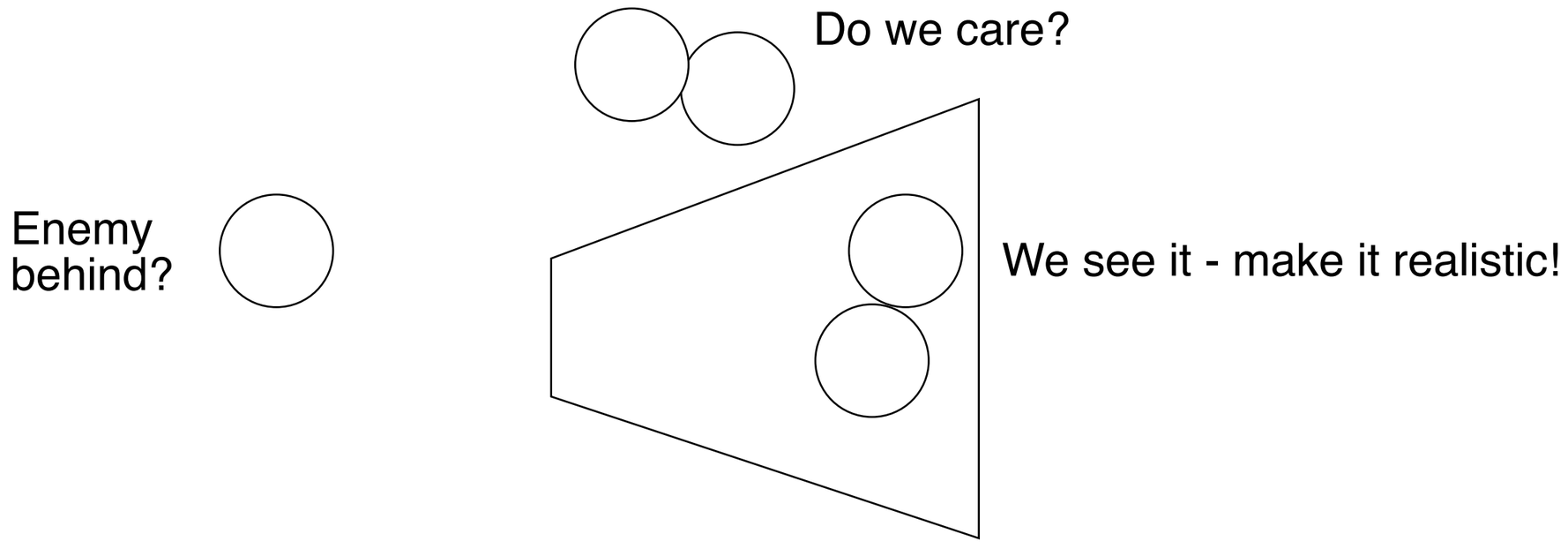
Subdivision by BSP trees, octrees, quad-trees,
regular grid.
Generally useful for most large world problems



Frustum culling and collision detection

If we can’t see it, do we care?

Frustum culling is already applied for drawing. Use it also for simplifying collision detection.



Collision detection and portals

Again, we can take advantage of visibility to reduce workload.

Portals lead to problems, objects near portals in cells and portals systems need to be present in both cells.

Process collision detection and AI in visible cells only, or visible and nearby.

Collision handling

What to do once a collision is found.

- Separate
- Change velocities

- Apply torque/rotations
- Deform
- Maintain constraints

Full (narrow-phase) tests are hard to resolve.

Simple cases for collision detection

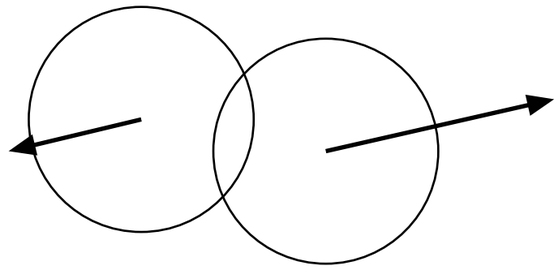
- Collisions between objects with same mass
- Collision with stationary (infinte mass) object
- No rotation (particles, spheres)

Separate

1) Intuitive: Immediately push objects from each other.
May add/change energy.

2) Better: Disregard collisions between objects moving away from each other.

Test with dot product of position and velocity difference.



These two have already collided!

Simple particle physics (again)

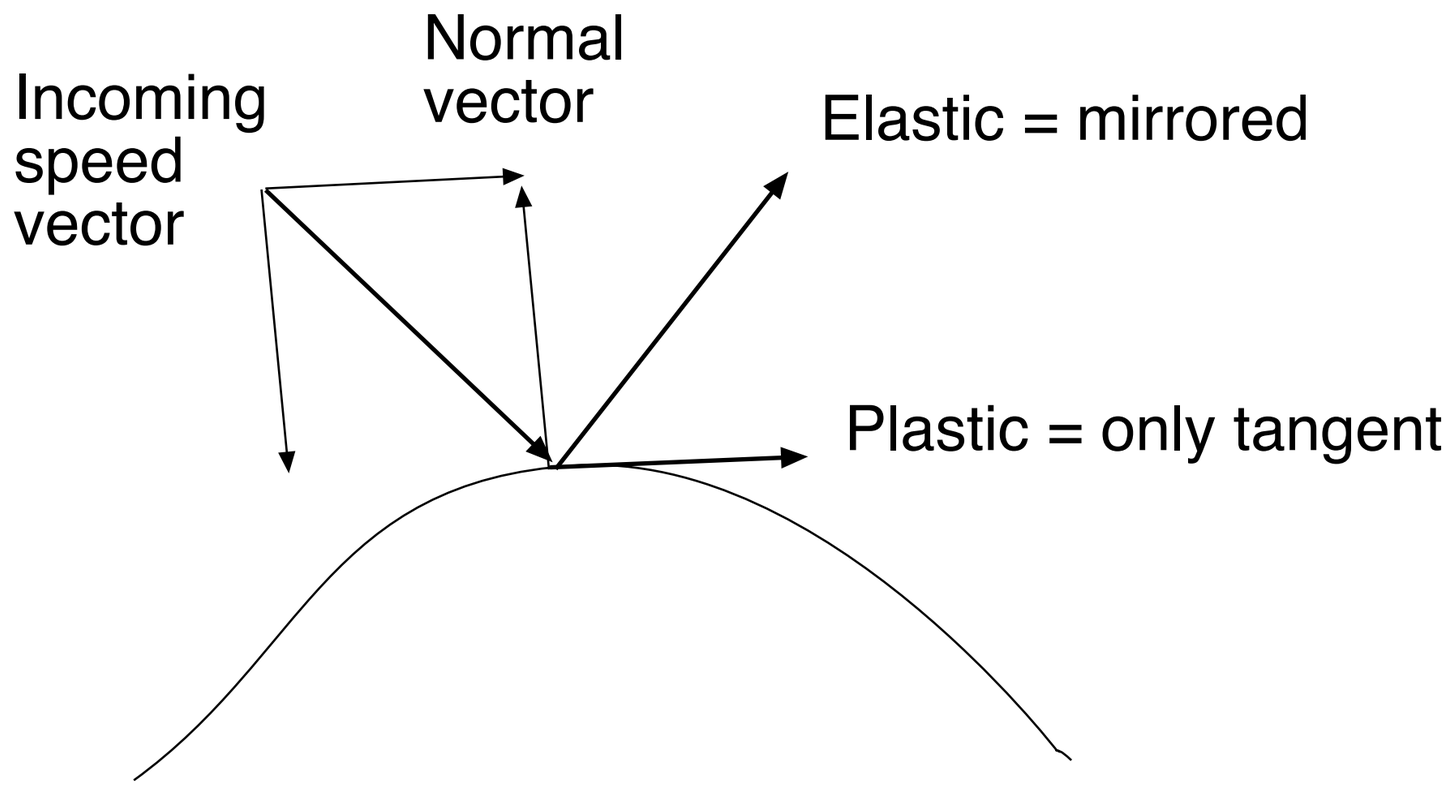
acceleration = gravity + forces/mass
speed = speed + acceleration
position = position + speed

$$a = (g + \sum f/m) \cdot \Delta t$$
$$s = s + a \cdot \Delta t$$
$$p = p + s \cdot \Delta t$$

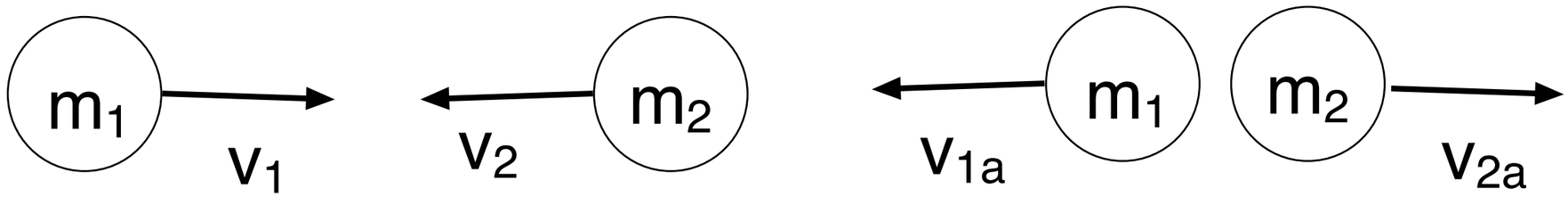
(“Euler integration”)

Modify speed and position in collisions

Simple particle-surface collision



Plastic and elastic collisions



Preserve momentum

$$m_1v_1 + m_2v_2 = m_1v_{1a} + m_2v_{2a}$$

Elastic collisions also preserve kinetic energy

$$m_1v_1^2 + m_2v_2^2 = m_1v_{1a}^2 + m_2v_{2a}^2$$

Plastic collisions

Two objects with different mass collide.

Very easy to solve:

$$v_{1b}m_1 + v_{2b}m_2 = v_a m_1 + v_a m_2 = v_a(m_1 + m_2)$$

$$v_a = (v_{1b}m_1 + v_{2b}m_2)/(m_1 + m_2)$$

Elastic collisions

Two objects with different mass collide.

Trickier to solve. Solution is:

$$v_{1a} = (v_{1b}(m_1 - m_2) + 2v_{2b}m_2)/(m_1 + m_2)$$

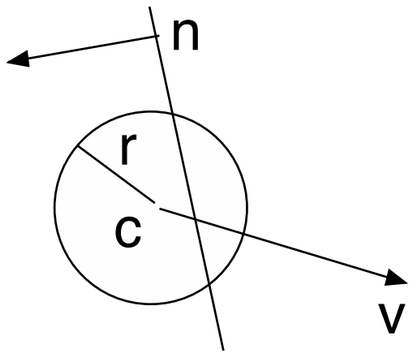
$$v_{2a} = (2v_{1b}m_1 + v_{2b}(m_2 - m_1))/(m_1 + m_2)$$

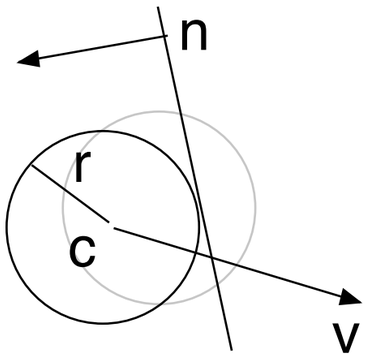
Add a restitution parameter:

$$v_{1a} = (v_{1b}(m_1 - \varepsilon m_2) + (1+\varepsilon)v_{2b}m_2)/(m_1 + m_2)$$

$$v_{2a} = ((1+\varepsilon)v_{1b}m_1 + v_{2b}(m_2 - \varepsilon m_1))/(m_1 + m_2)$$

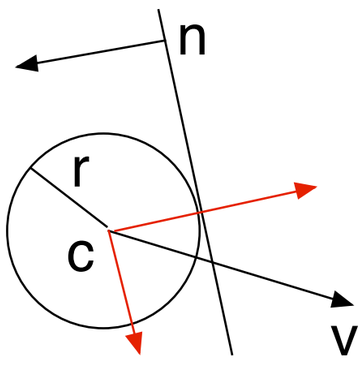
Collision handling sphere-polyhedra



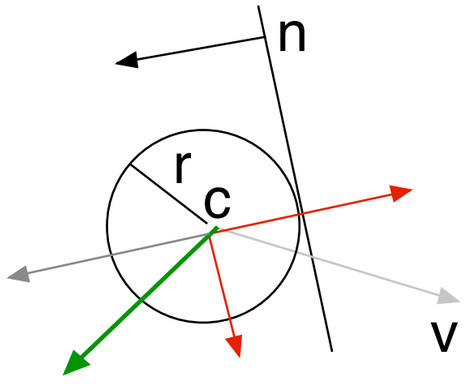


Assuming stationary polyhedra

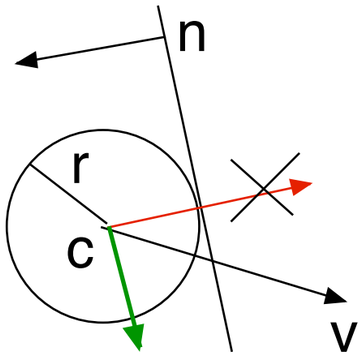
Want to separate? Move object away along normal vector



Split velocity along n

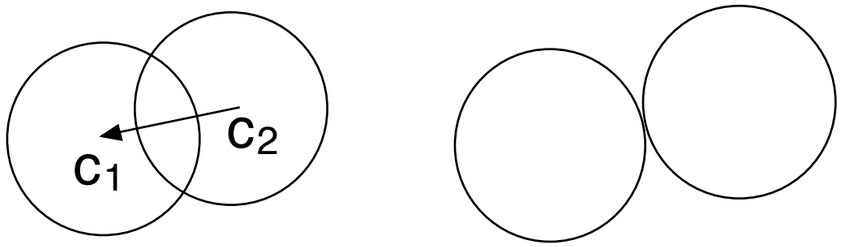


Elastic

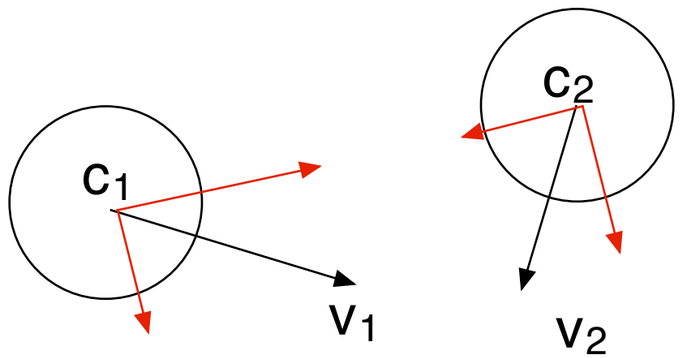


Plastic

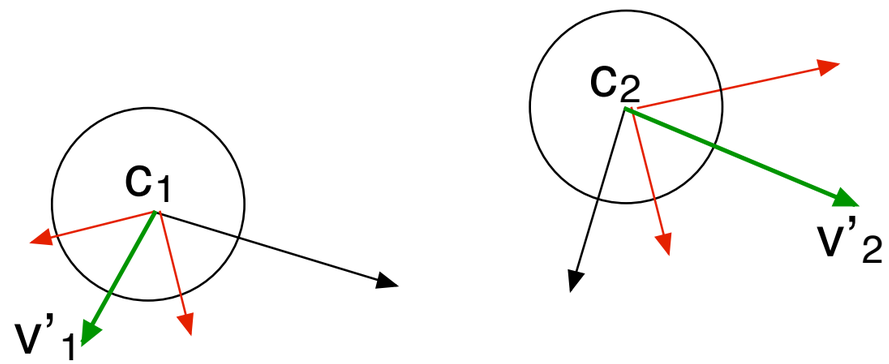
Collision handling sphere-sphere
(simplified, point masses)



Want to separate? Move object away along vector through centers



Split velocities along vector though centers (c_2-c_1)



Elastic: Exchange components along c_2-c_1

Beyond point-mass mechanics

Rigid body mechanics
Rotations
Better integration
Stacking
Applying forces and backing time to avoid overlap
Deformable bodies
Breakable bodies

Supplement on collision detection and handling

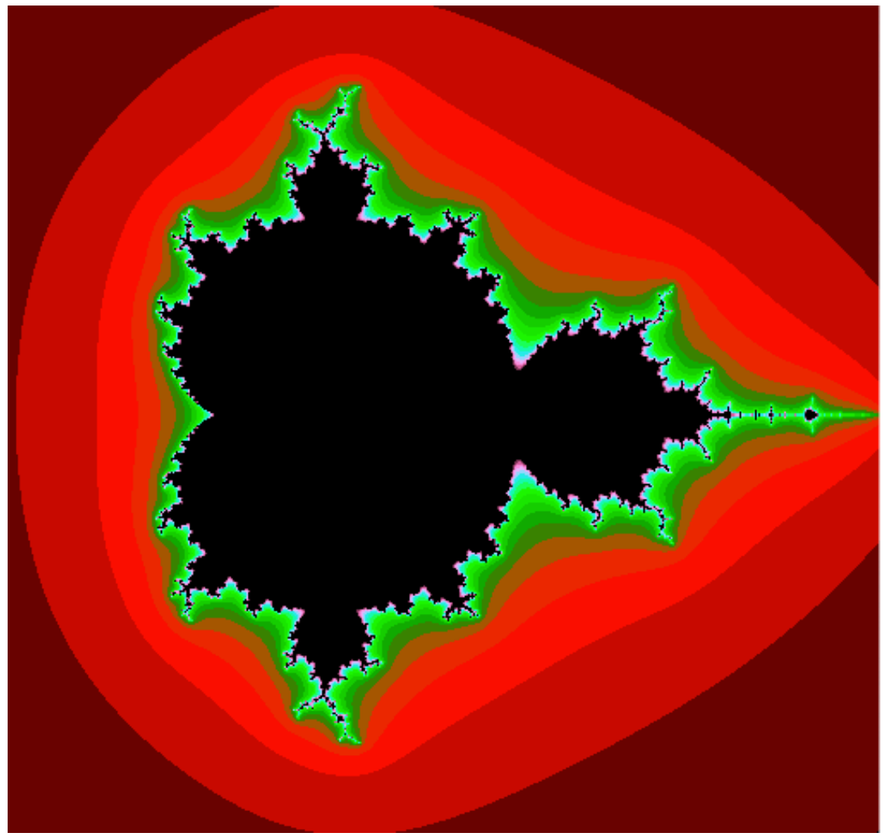
Some points above were corrections/clarifications over the book and there should be a text about it.

The supplement is uploaded to the course page in its first, preliminary form.

Fractals

Creating complex and interesting shapes from code

Most famous fractal: Mandelbrot set

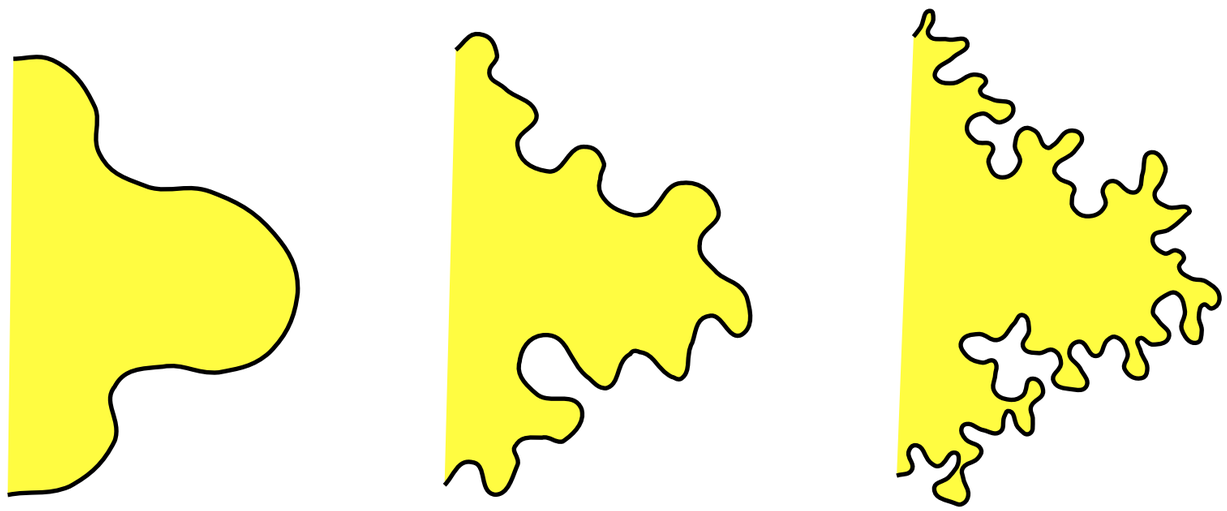


What is it, more than a pretty image?

Natural objects have fractal features

Classic example: Coastline

Shape and length varies with resolution



Classic example 2: Bracken

Self-similar, variable scale

Fractals in computer graphics

Fractals are shapes with:

- self-similarity
- infinite resolution

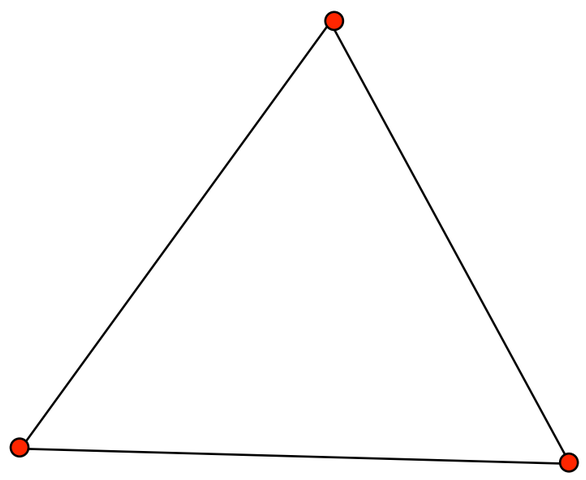
Used for modelling such shapes

Classification of fractals

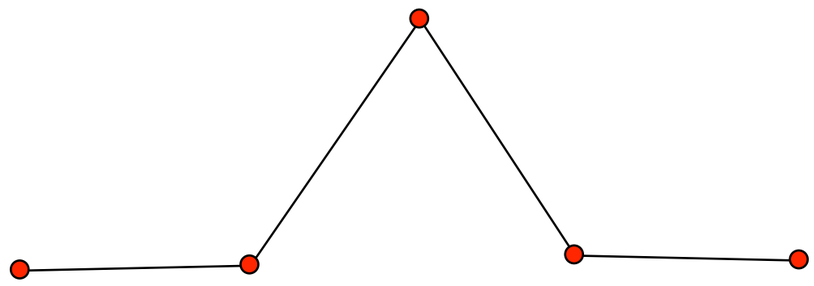
- geometrical recursive construction
- stochastic fractals
- mathematical formulas (in the complex plane)

Geometric construction of self-similar fractals

Example: Koch curve

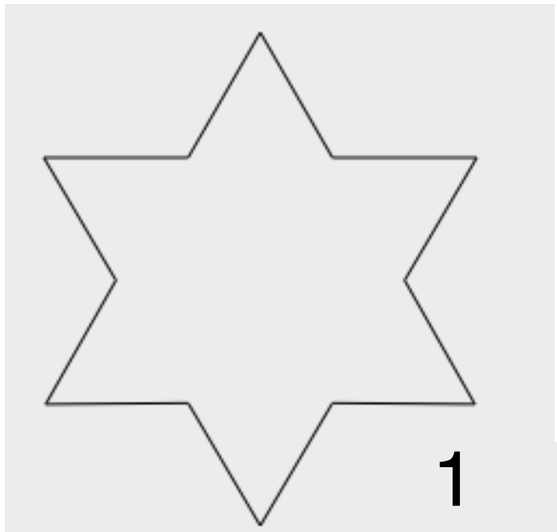


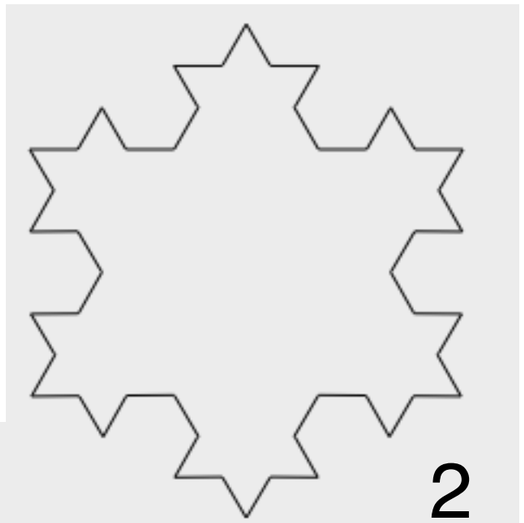
Initiator

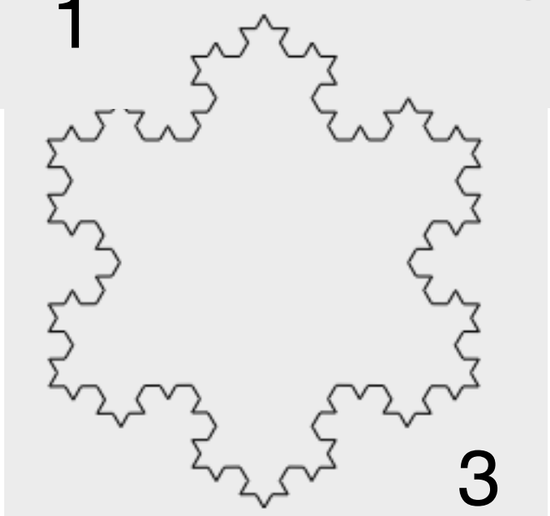


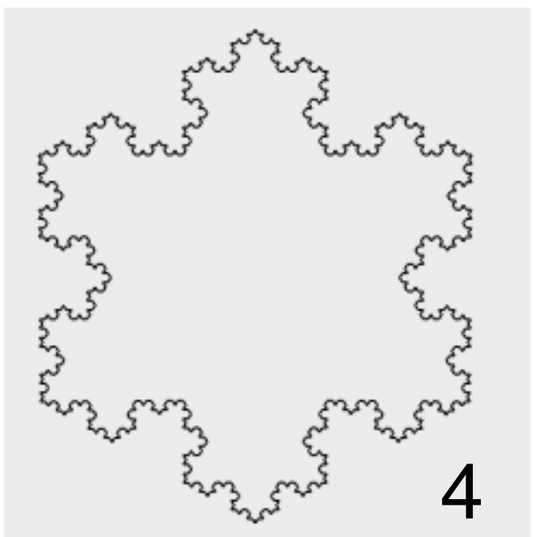
Generator

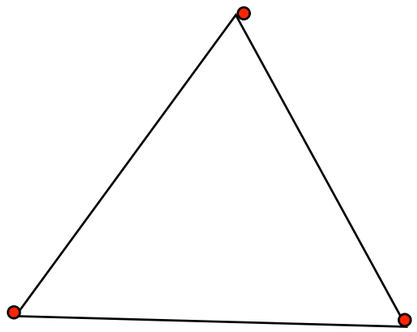
Resulting Koch curves



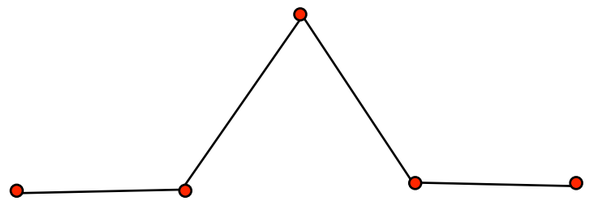








Initiator



Generator

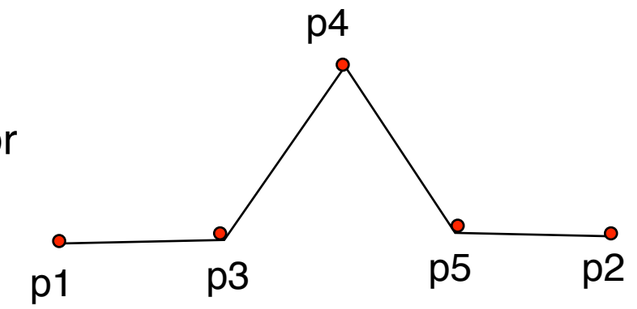
Recursive function

Pass all parts to next level

Replace part with the generator, *scaled to same length*

Stop at desired recursion depth or when sections are small enough (e.g. 1 pixel long)

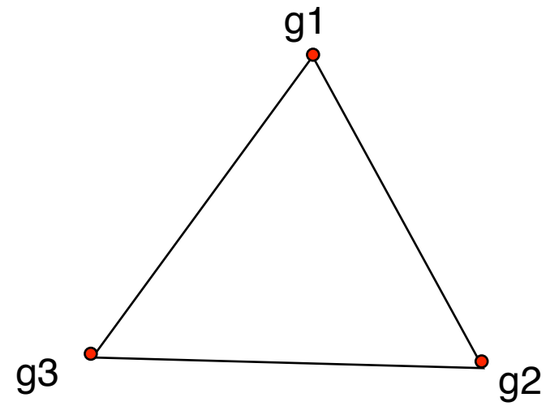
```
procedure DrawKoch(p1, p2, depth)

if depth >= maxDepth then

    MoveTo(p1)
    LineTo(p2)
    return

else

    calculate p3, p4, p5 as the three points inside the generator

    DrawKoch(p1, p3, depth+1)
    DrawKoch(p3, p4, depth+1)
    DrawKoch(p4, p5, depth+1)
    DrawKoch(p5, p2, depth+1)
```



main procedure:

Choose three generator points, g1, g2, g3

```
DrawKoch(g1, g2, 0)
DrawKoch(g2, g3, 0)
DrawKoch(g3, g1, 0)
```



Fractal dimension

A measure of how rough or fragmented the shape is

Definition:

$$ns^D = 1$$

n = number of subparts

s = scaling

D = fractal dimension

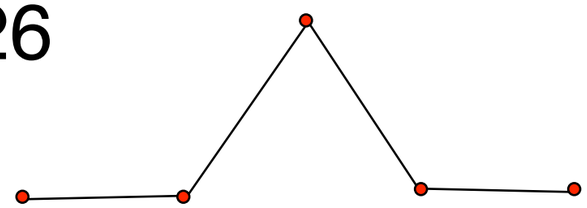
Solves to D = ln(n) / ln (1/s)

Fractal dimension example: Koch curve

n = 4

s = 1/3

D = ln 4 / ln 3 = 1.26



Fractal dimension example: Splitting a line

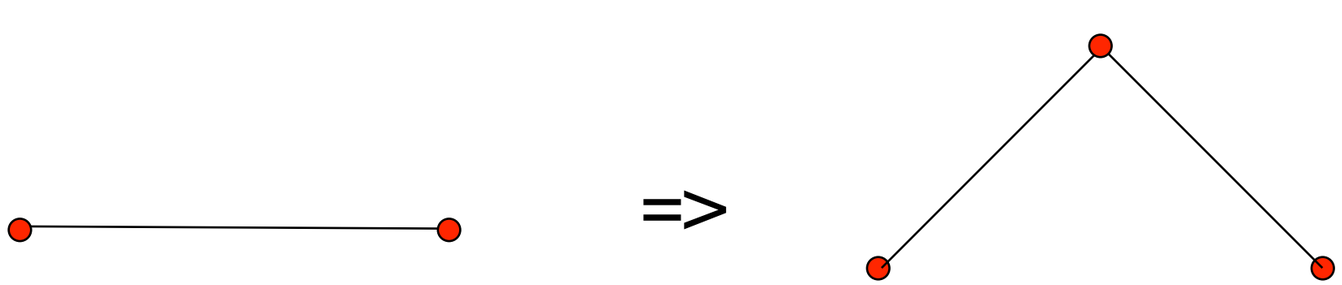
=>

n = 2

s = 1/2

D = ln 2 / ln 2 = 1

Fractal dimension example: Splitting a line and moving midpoint



$$n = 2$$

$$s = 1/\sqrt{2}$$

$$D = \ln 2 / \ln \sqrt{2} = 2$$

Fractal dimension:

In 2D:

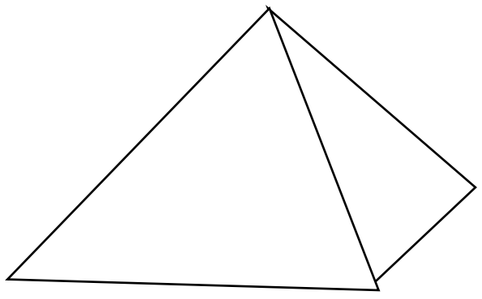
1 to 2: Well-behaved fractal curve

>2: Self-intersecting, area-covering

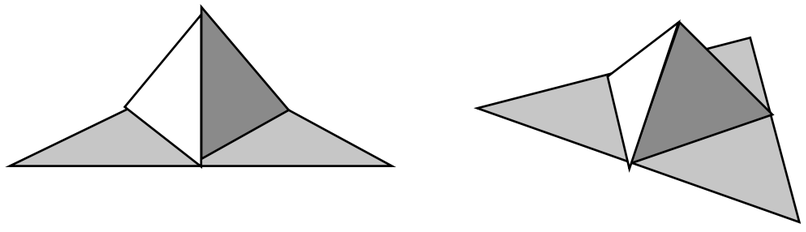
Split line: D = 1 minimum, no fractal
Koch: D = 1.26, moderate fractal
Moved midpoint: D = 2, maximum

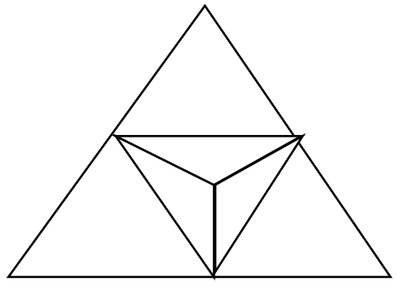
Geometric construction of self-similar fractals in 3D

Example



Initiator





Generator

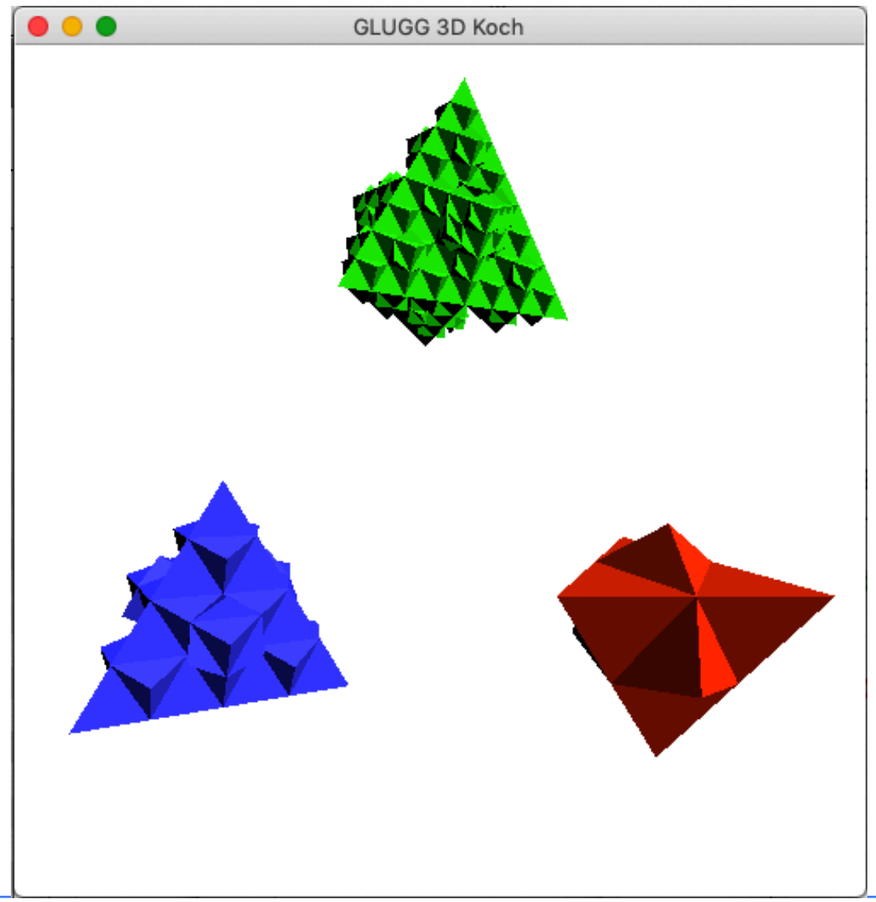
n = 6

s = 1/2

D = ln 6 / ln 2 = 2.58

Koch in 3D

Subdivide triangles, built using GLUGG



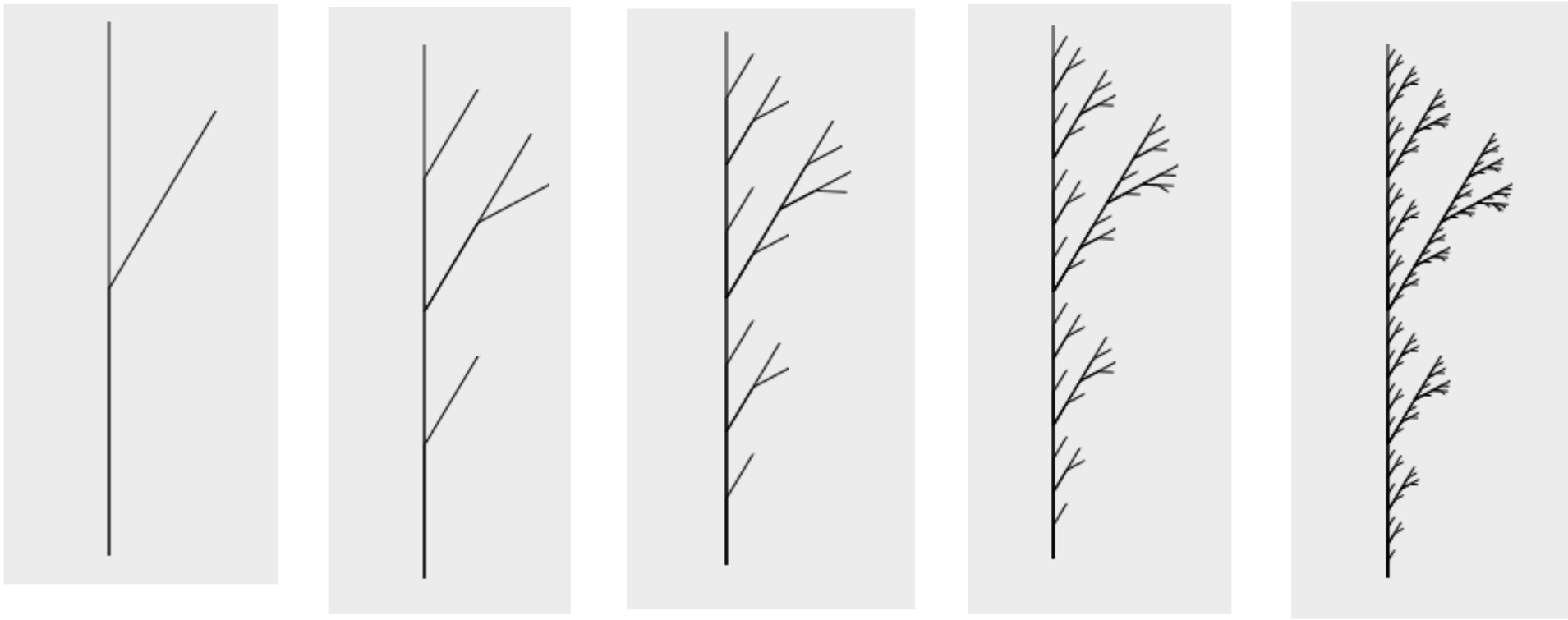
Interpretation of fractal dimension:

In 3D:

2 to 3: Well-behaved fractal surface

>3: Self-intersecting, volume-covering

Example: Generation of plants



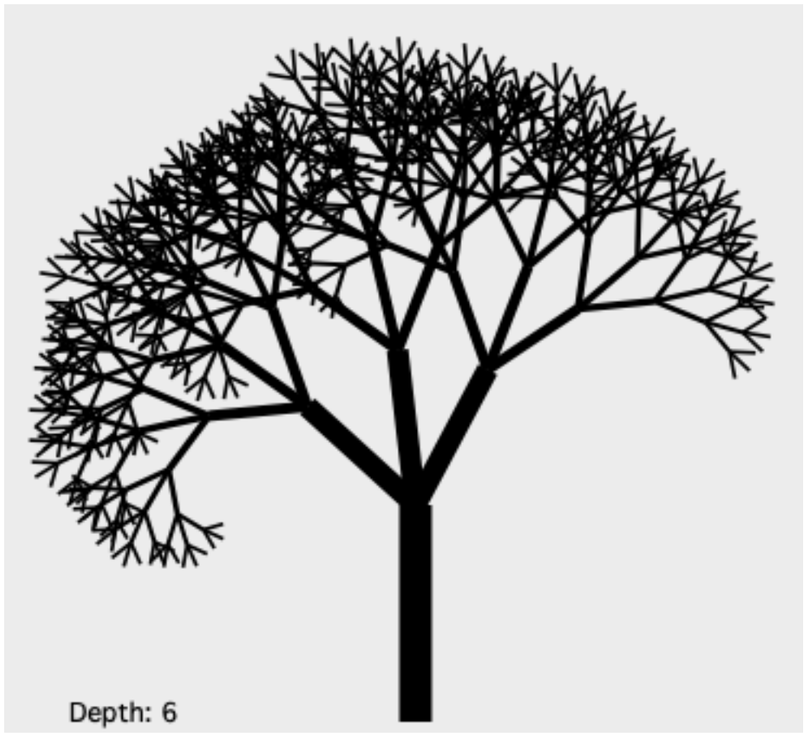
Promising, but too self-similar!

Statistically self-similar fractals

Random variation of generator

Same branch generator as before, with some or more randomness!

Example: Generation of plants #2



The bracken ("Barnsley fern"): Geometric fractal, iterated function system, IFS.

4 sets of transformations (combinations of rotation, scale, translate)
Recursions are here replaced by randomness.

Related methods:

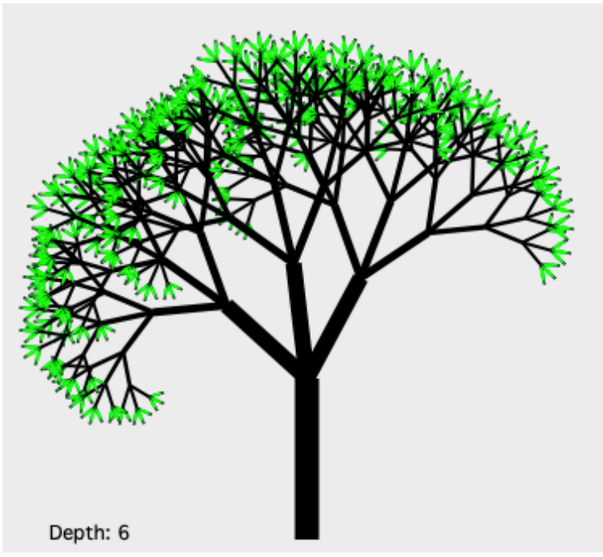
Shape grammars and procedural methods

No unlimited resolution

Different rules at different levels

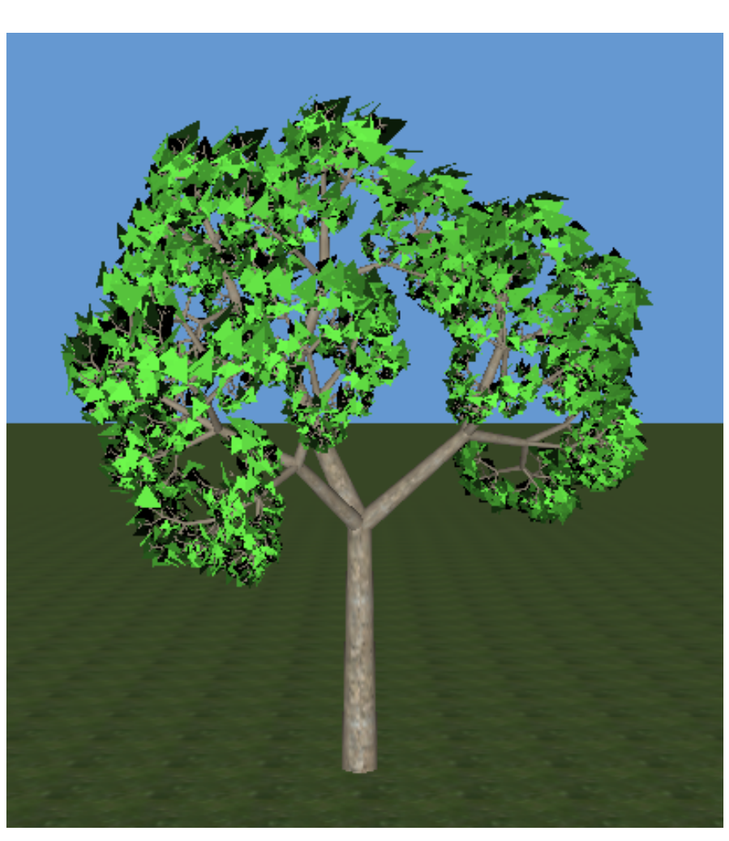
Example: Tree with leaves: replace last iteration with leaf generator

“graftals”



Tree in 3D

Lab in another course. Get the trunk, generate, scale and move.



Self-squaring fractals

Based on simple functions in complex space

Insert complex numbers (points) into a function

Apply function recursively, and analyze the behaviour.

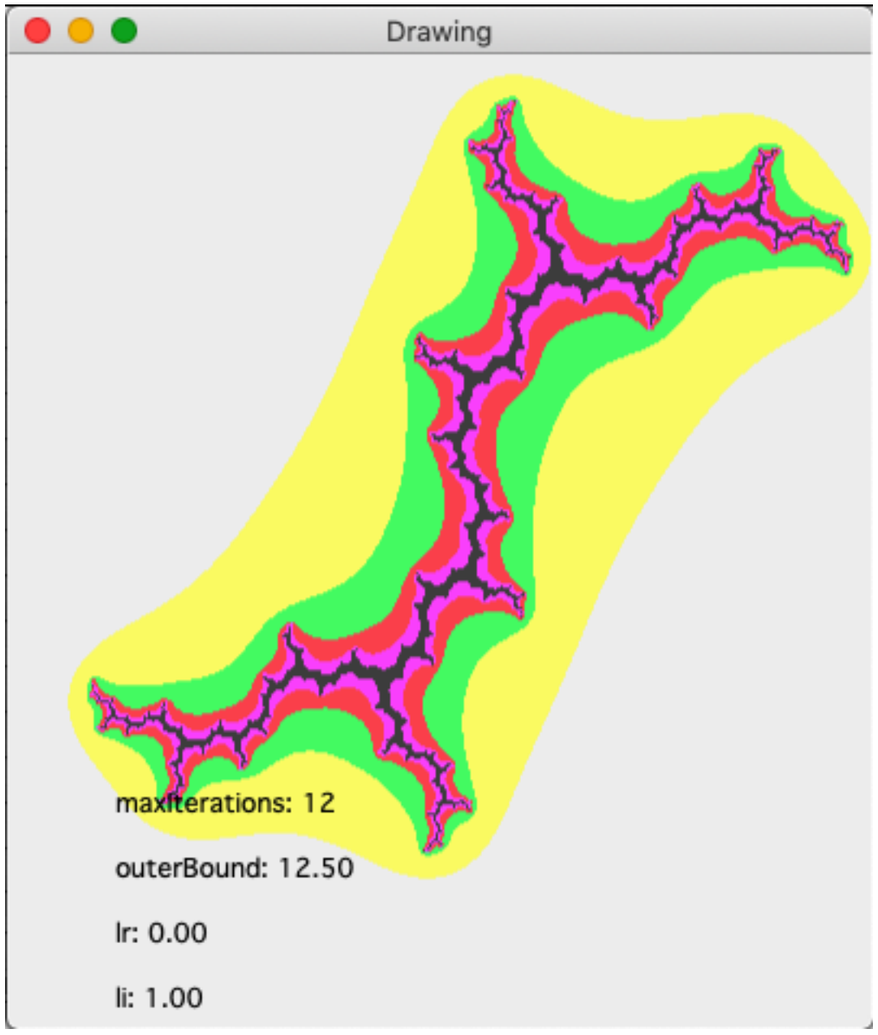
- Diverge?
- Converge?
- Chaotic?

Converge or chaotic: Does it keep within some limit in a number of iterations?

Self-squaring fractals

The Julia set

$$z_{k+1} = z_k^2 + \lambda$$



Julia set for $\lambda = (0, 1) = 0 + j$

The Julia set - Implementation

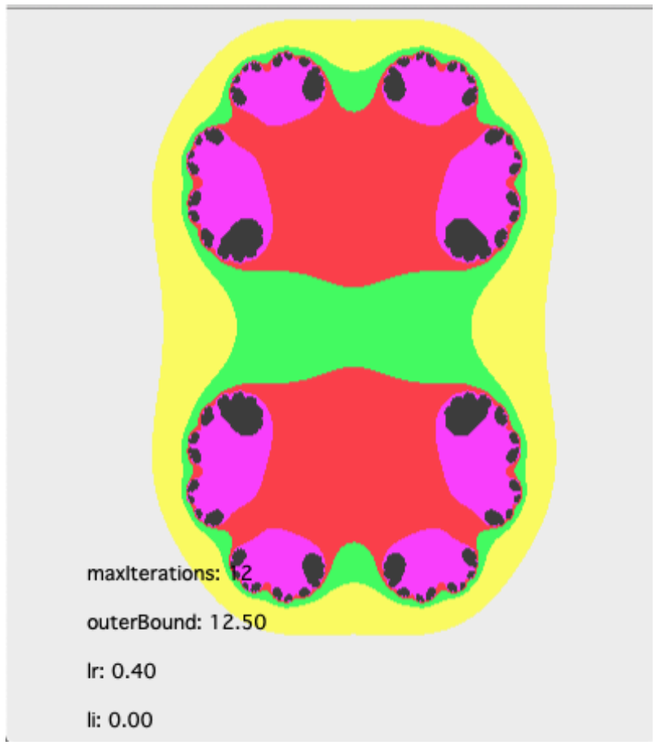
```
for y = miny to maxy
   for x = minx to maxx
      (zr, zi) = scaling of (x,y)

for i = 0 to maxiterations
   z = z² + λ
   if |z| > R then Leave

Draw pixel (x,y) (different colors for different i)
```

maxiterations ≈ 15 enough for decent result.

$R^2 \approx 10$

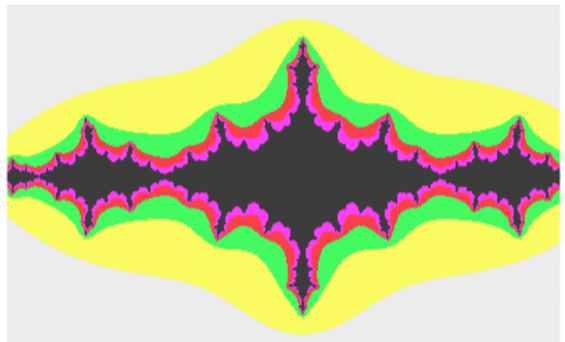
Other Julia sets

$$z_{k+1} = z_k^2 + \lambda$$

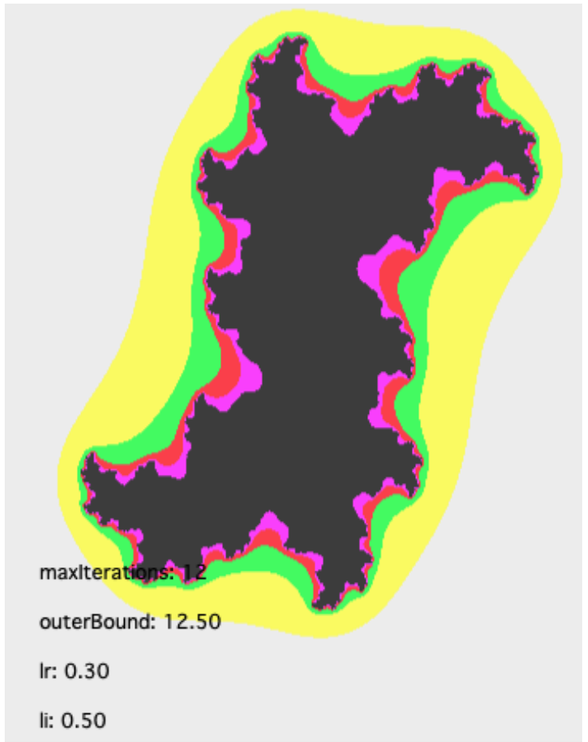


$\lambda = (0.4, 0)$

Other λ values



$\lambda = (-1.3, 0)$

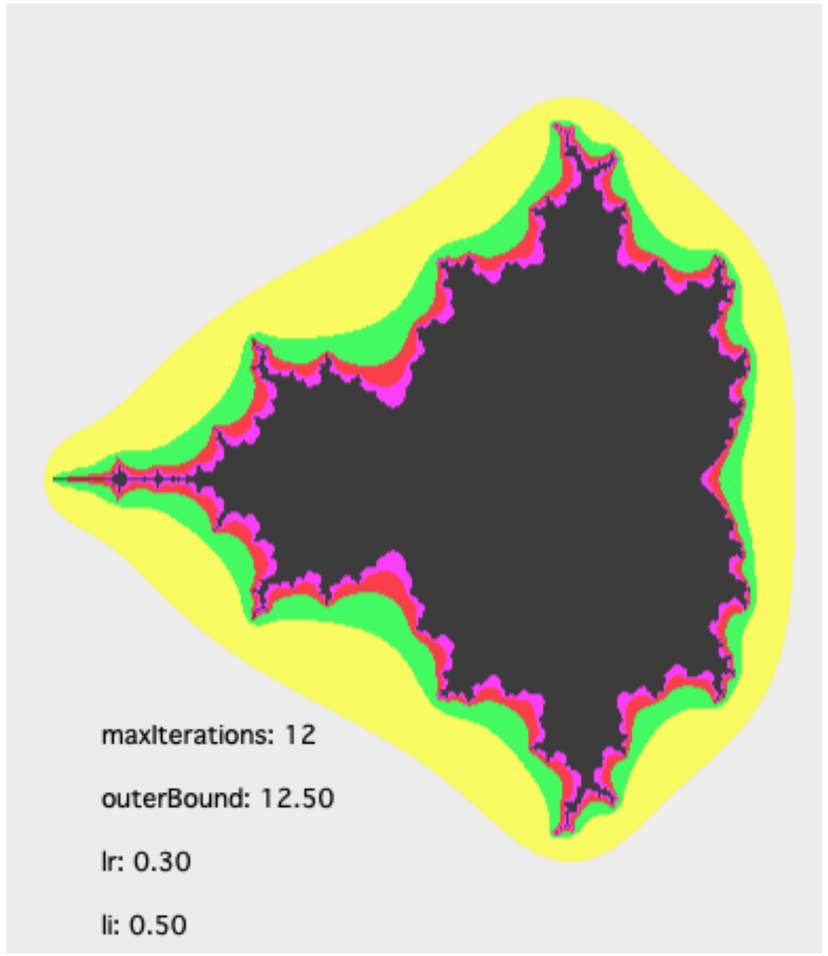


$\lambda = (0.3, 0.5)$

Self-squaring fractals

The Mandelbrot

$$z_{k+1} = z_k^2 + z_0$$



More 3D fractals

Mandelbulb. Based on polar coordinates rather than complex numbers.

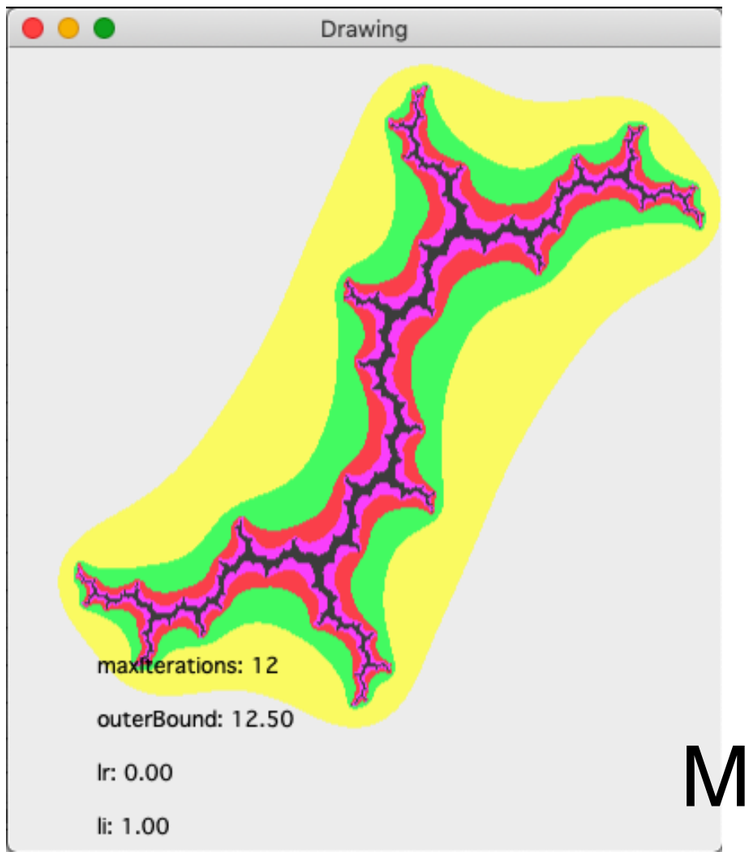
IMAGE REMOVED

Mandelbulb

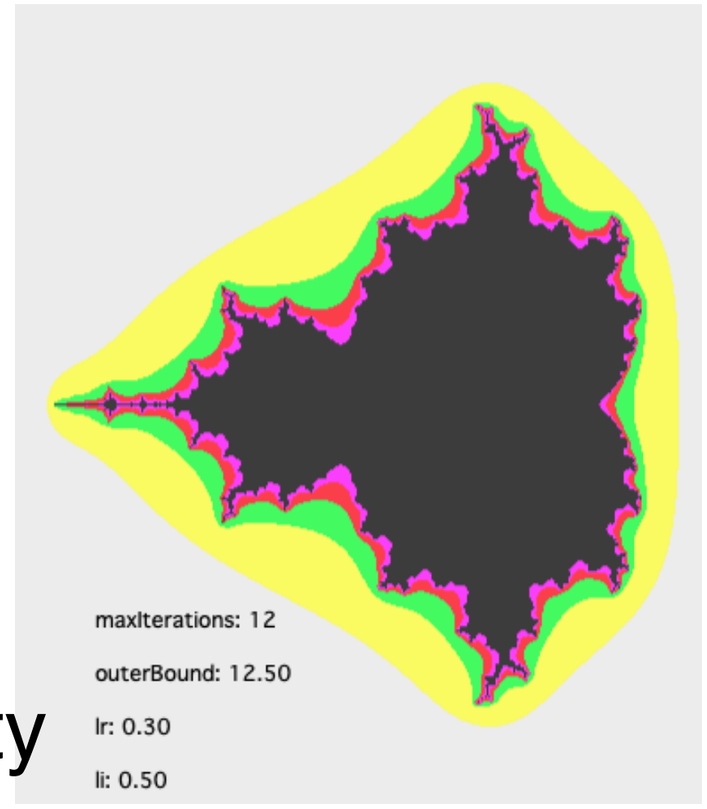
Several different variations. Amazing surrealistic scenes! Some potentially useful - but you will rather adapt yourself to the fractal than the fractal to you needs.

Many other 3D fractals exist.

Self-squaring fractals



- Beautiful
- Non-predictable
- Limited usability



Mathematical curiosity

Fractals, summary

1) Geometrically constructed fractals

Very useful for generating many kinds of natural objects

Allows design of complex models with arbitrary resolution

2) Self-squaring fractals (and other adventures in the complex plane)

Questionable practical usability

Hard to do planned designing

Next time

Fractal terrains

Anti-aliasing

Raytracing