No Blender Installed?

Find a buddy!

Pair up & make a friend!

Much Visual, Very 3D, Many Python.

Vincent D. Warmerdam, GoDataDriven, @fishnets88 - koaning.io

vincent: whois

my blender story

Today

We will say no to **Clicky-Clicky** and say yes to **Typy-Typy**.

We will generate a lot of cubes, make pretty renders and even animate some cubic explosions. We will do this by following python examples and lots of hacking freedom.

We will spend some time talking about the UI and then we will only talk about code. You will code more than I will talk.

You will need a recent version of blender, a text editor and creativity. Try to pair up if this is a problem!

Today

- explain basic blender ui (15 min max!)
- show how to do code in blender
- get creative with cubes
- introduction to recursive art
- maths and trees
- advanced recursive art patterns
- animation + physics
- rendering with blender code

Some Basic Movement

Try the following:

- moving with scrollwheel
- **CRTL** + scrollwheel
- **SHIFT** + scrollwheel

Basic editing

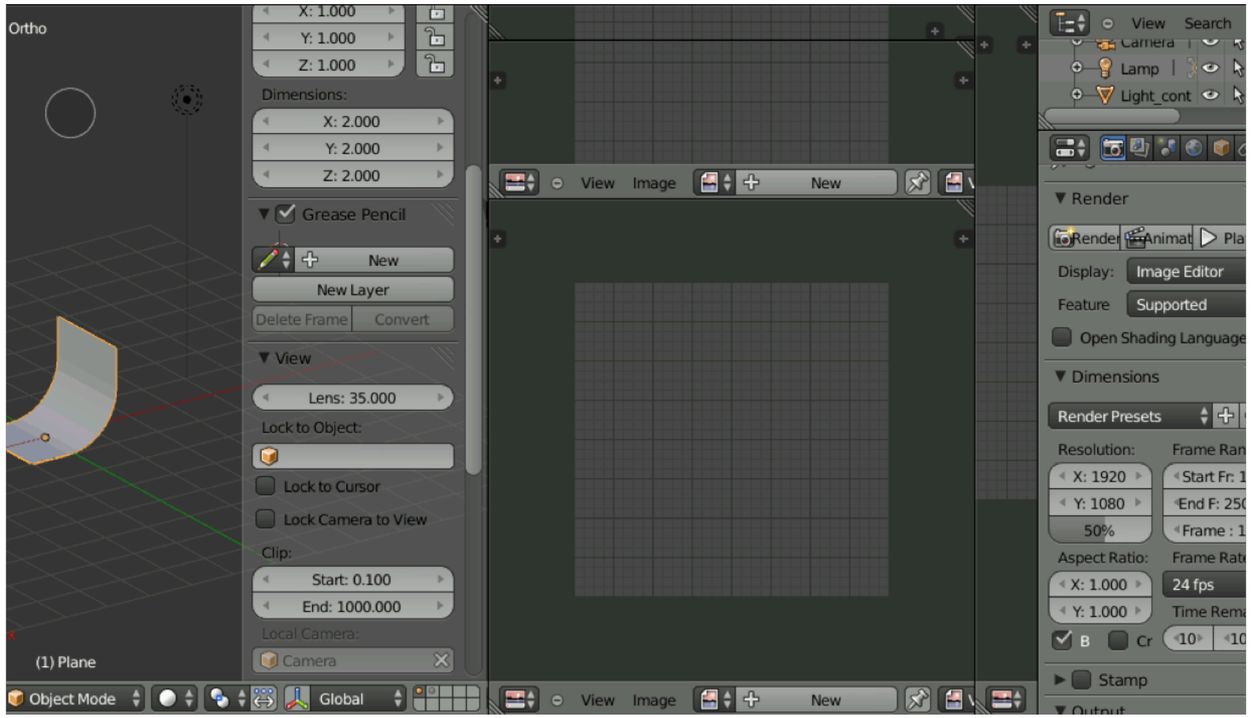
What do you think the **a**, **s**, **r** and **g** buttons do?

Useful

- **ALT + F**: quick view change
- **RMB**: select a single object in the view
- **MMB**: rotation
- **space**: quicktype shortcut

Blender UI Got-Ya's

We will be opening View windows. Closing them is an annoying UI-lesson. Thankfully, we have blender.stackexchange to the rescue.



Assignment!

Add some cubes.

You can do this by finding the appropriate button or via **SPACE**.

Why code 3D

Can't we sculpt by hand?

Counter argument: procedural generation.

Button = Code

This is something that makes blender very cool.

Every button in Blender runs code.

If you want the code that is run, just hover your mouse over it and you'll see the code.

To share code

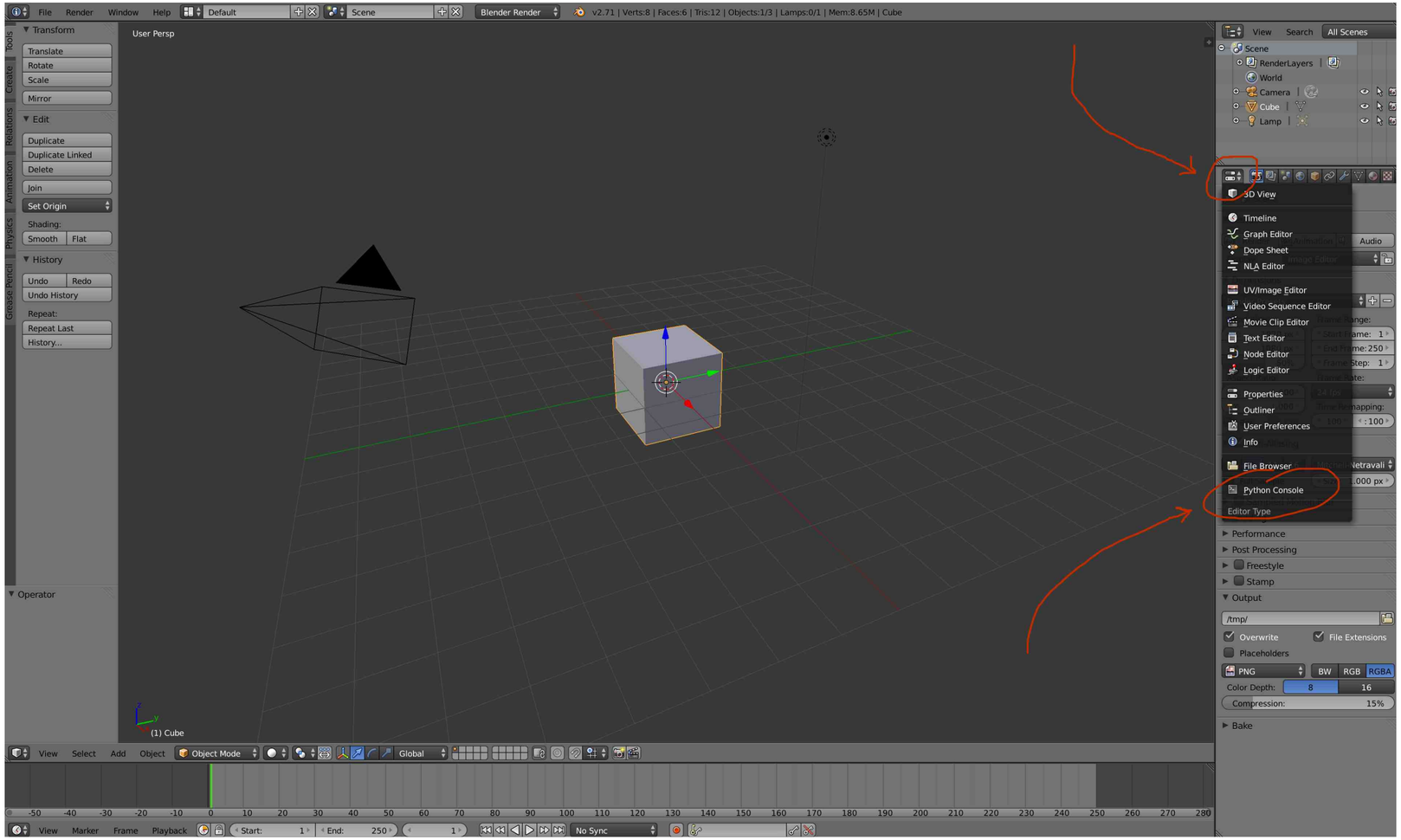
It will be useful for us to share code.

Please go to this website.

`https://codeshare.io/KHDue`

I will paste all my code there, please do not delete it.

Python Console



How today will work.

We will write scripts that generate scenes and run them in Blender.

We will only do basic things today, mainly with cubes. Once you get the hang of it you can get creative with things.

How to run a python script from blender;

```
filename = "some_path/file.py"
exec(compile(open(filename).read(), filename, 'exec'))
```

Note: to autocomplete in the terminal press **CTRL+SPACE**, not TAB.

What does this code do?

```
bpy.ops.mesh.primitive_cube_add(radius=1, location = (0,0,0))
```

What does this code do?

```
bpy.ops.mesh.primitive_cube_add(radius=4, location = (0,0,0))
bpy.ops.mesh.primitive_cube_add(radius=3, location = (10,0,0))
bpy.ops.mesh.primitive_cube_add(radius=2, location = (20,0,0))
bpy.ops.mesh.primitive_cube_add(radius=1, location = (30,0,0))
```

Dude, Remove these!

```
def delete_all():
    bpy.ops.object.select_all(action='SELECT')
    bpy.ops.object.delete(use_global=True)

delete_all()
```

What does this code do?

```
numcubes = 6
rcubes = 0.3
for x in range(numcubes):
    for y in range(numcubes):
        for z in range(numcubes):
            bpy.ops.mesh.primitive_cube_add(
                radius=rcubes, location = (x,y,z)
            )
```

What does this code do?

```
import math

def f(x,y):
    return 5*sin(x/15.0*math.pi) + 5*cos(y/15.0*math.pi)


numcubes = 8

for x in range(-numcubes,numcubes):
    for y in range(-numcubes,numcubes):
        bpy.ops.mesh.primitive_cube_add(
            radius=0.2, location = (x,y,f(x,y))
        )
```

Assignment, 15 mins.

Use the math library and create a cool math plot.

Edit Cubes

We'll code things just like how you would manually click things. After creating an object you can edit it's properties.

```
import numpy as np


c = bpy.data.objects["Cube"]
c.scale = (2,1,1)
c.rotation_euler = (np.pi/4, np.pi/4, np.pi/4)
c.location = (1,2,3)
```

There are many other properties to set like material but we'll get to those later.

Hindsight

If you know beforehand what you'll draw then this should work. If you don't then you may want to edit things in hindsight.

```
for item in bpy.data.objects:
    print(item)
```

Alternatively you can also edit properties as you're creating objects.

Something cool: Recursion!

Do we know what recursion is?

Assignment: Clean up this code and play!

```
def new_cube(old_loc, direction, rad, dimmer):
    res = []
    for i in [0,1,2]:
        res.append(old_loc[i] + direction[i]*dimmer +  2 * direction[i]*rad )
    return [rad*dimmer,res]

def rec(cube, depth):
    if depth == 4 :
        return None
    else:
        bpy.ops.mesh.primitive_cube_add(
            radius=cube[0], location = cube[1]
        )
        print(cube)
        rec( new_cube(cube[1],(1,0,0),cube[0],0.4) , depth + 1 )
        rec( new_cube(cube[1],(0,1,0),cube[0],0.4) , depth + 1 )
        rec( new_cube(cube[1],(0,0,1),cube[0],0.4) , depth + 1 )
        rec( new_cube(cube[1],(-1,0,0),cube[0],0.4) , depth + 1 )
        rec( new_cube(cube[1],(0,-1,0),cube[0],0.4) , depth + 1 )
        rec( new_cube(cube[1],(0,0,-1),cube[0],0.4) , depth + 1 )

rec([1,(0,0,0)],0)
```

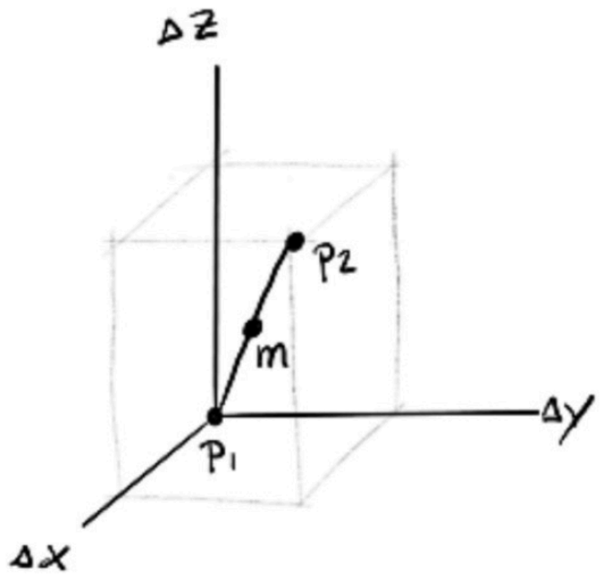
What does this code do?

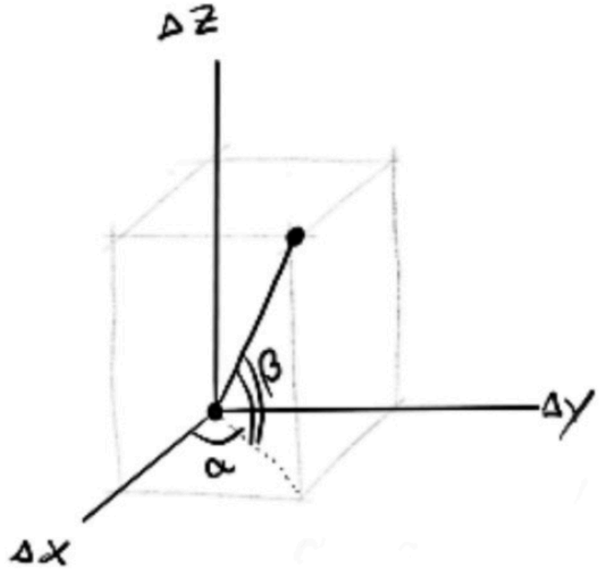
```
def randdir():
    choices = [(1,0,0),(0,1,0),(0,0,1),(-1,0,0),(0,-1,0),(0,0,-1)]
    return random.choice(choices)

def new_cube(old_loc, direction):
    res = []
    for i in [0,1,2]:
        res.append(old_loc[i] + direction[i])
    return tuple(res)

cube = (0,0,0)
for i in range(500):
    cube = new_cube(cube,randdir())
    bpy.ops.mesh.primitive_cube_add(radius=0.5, location = cube)
```

Best Recursive Example

How to build this in Blender

The hard part is that you'll need a bit of math to create an appropriate helper function.





How to build this in Blender

$$m = \frac{p_1 + p_2}{2}$$

$$L = \sqrt{\delta x^2 + \delta y^2 + \delta z^2}$$

$$\alpha = \arctan(\frac{\delta x}{\delta y})$$

$$\beta = \arctan(\frac{\delta z}{\delta L})$$

What does this code do?

```
def cylinder_between(x1, y1, z1, x2, y2, z2, r):
    dx = x2 - x1
    dy = y2 - y1
    dz = z2 - z1
    dist = np.sqrt(dx**2 + dy**2 + dz**2)
    bpy.ops.mesh.primitive_cylinder_add(
        radius = r,
        depth = dist,
        location = (dx/2 + x1, dy/2 + y1, dz/2 + z1)
    )
    phi = np.arctan2(dy, dx)
    theta = np.arccos(dz/dist)
    bpy.context.object.rotation_euler[1] = theta
    bpy.context.object.rotation_euler[2] = phi
```

What does this code do?

```
def branch(origin, depth = 1):
    if depth > 10:
        return 0
    x,y,z = origin
    x_new = x + np.random.normal(0, 2, 1)
    y_new = y + np.random.normal(0, 2, 1)
    z_new = z + np.random.uniform(2, 10, 1)
    cylinder_between(x,y,z,x_new, y_new, z_new, r = 1)
    if np.random.random() < 0.4:
        branch((x_new, y_new, z_new), depth = depth + 1)
    branch((x_new, y_new, z_new), depth = depth + 1)
```

Assignment

Wonder what the downside of this is while Vincent uploads the code.

```
def branch(origin, depth = 1):
    if depth > 10:
        return 0
    x,y,z = origin
    x_new = x + np.random.normal(0, 2, 1)
    y_new = y + np.random.normal(0, 2, 1)
    z_new = z + np.random.uniform(2, 10, 1)
    cylinder_between(x,y,z,x_new, y_new, z_new, r = 1)
    if np.random.random() < 0.4:
        branch((x_new, y_new, z_new), depth = depth + 1)
    branch((x_new, y_new, z_new), depth = depth + 1)
```

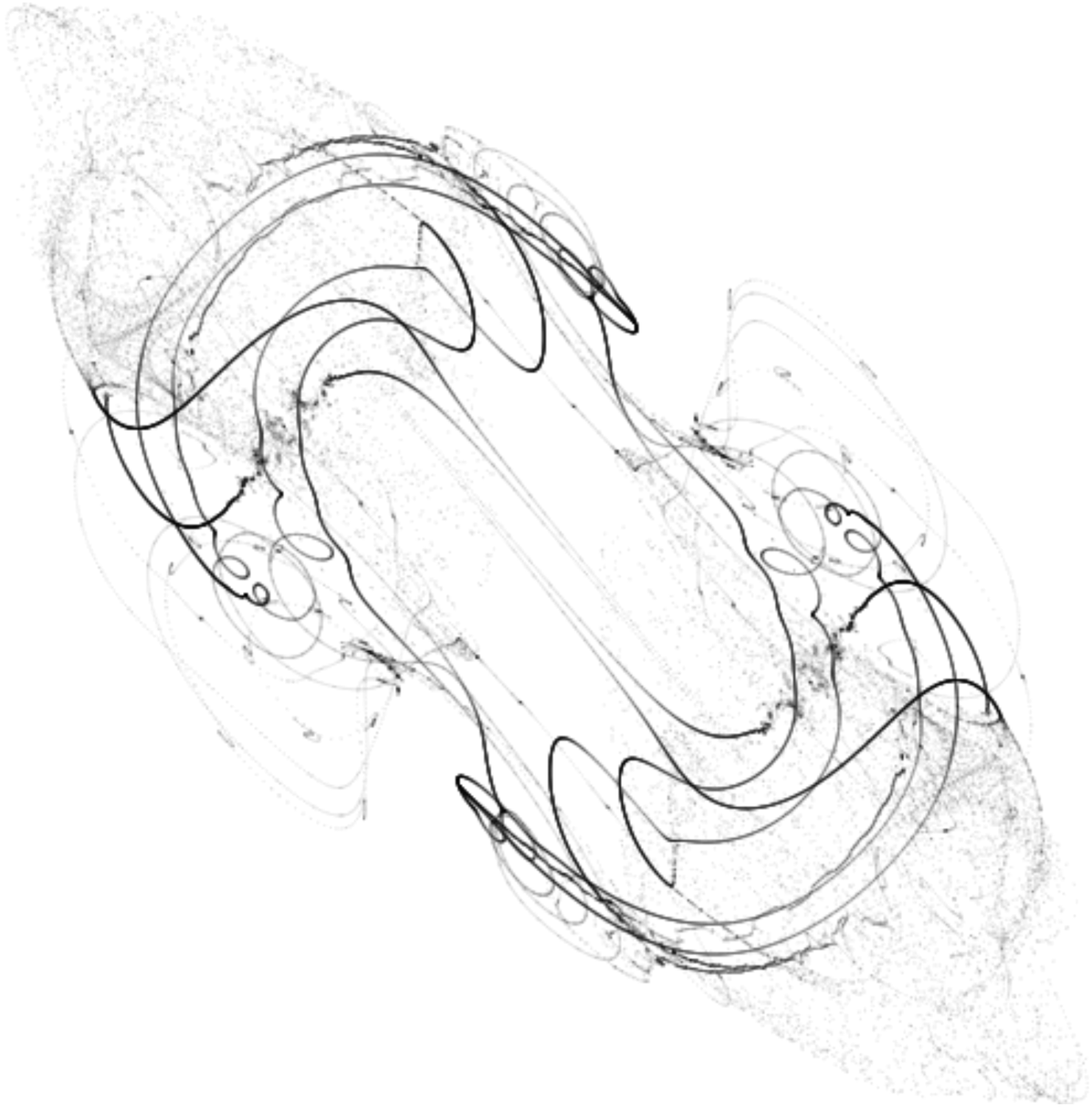
Alternative Recursion

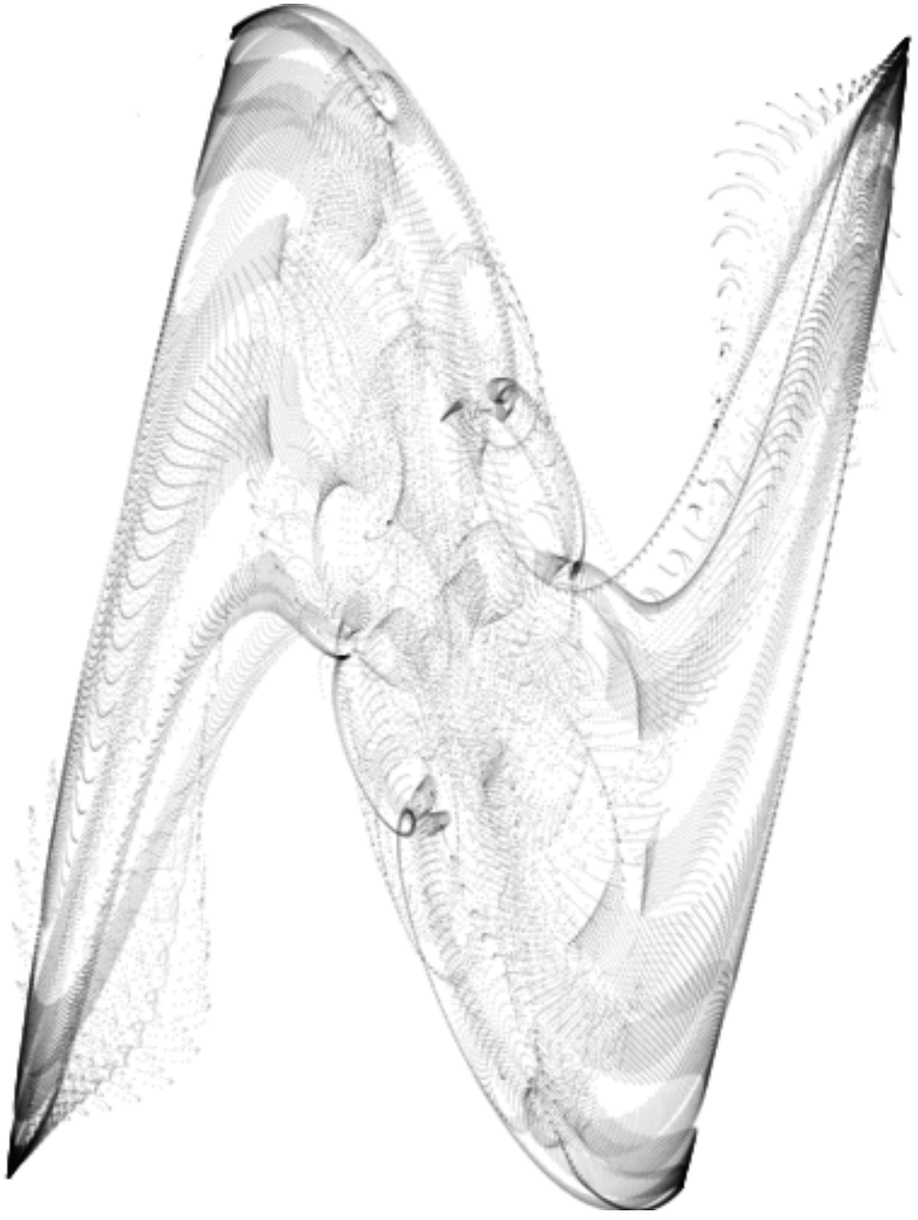
Let's define how points are drawn in a two dimensional system.

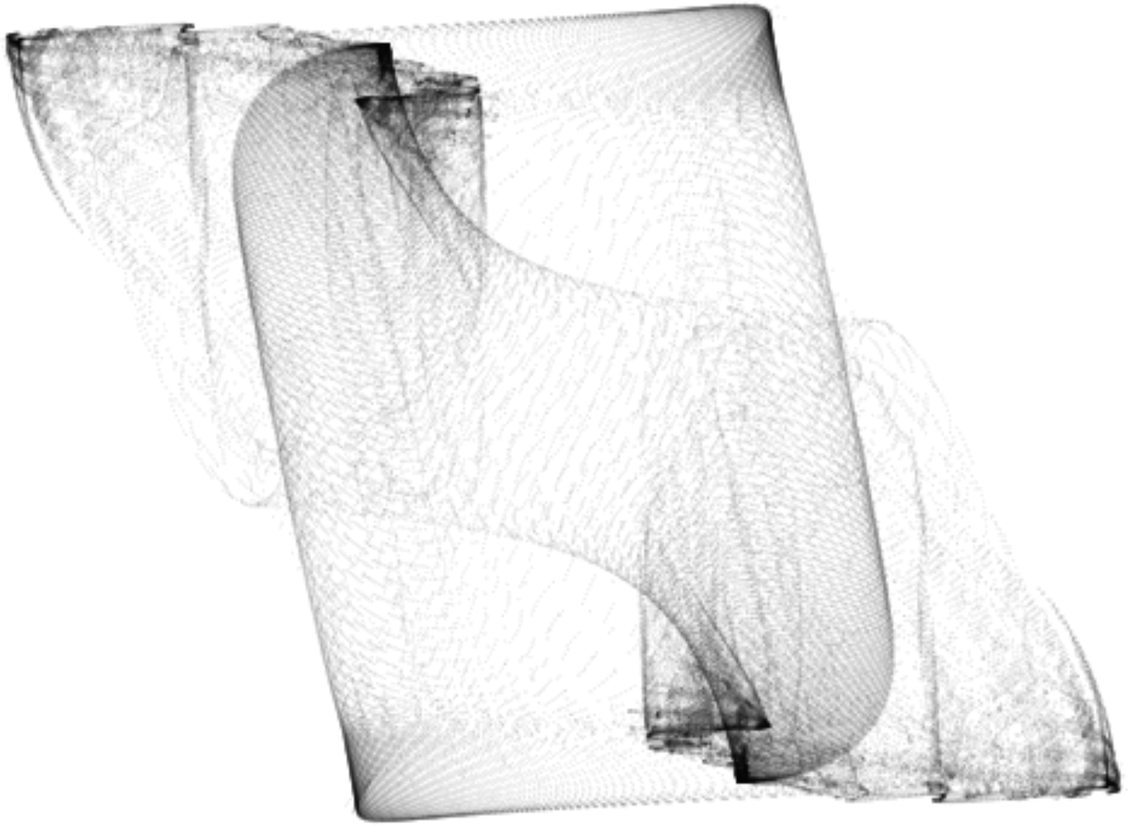
$$x_{n+1} = \sin(a \times y_n) + \cos(b \times x_n) - \cos(c \times z_n)$$
$$y_{n+1} = \sin(d \times x_n) + \cos(e \times y_n) - \cos(f \times z_n)$$
$$z_{n+1} = z_n + 0.1$$

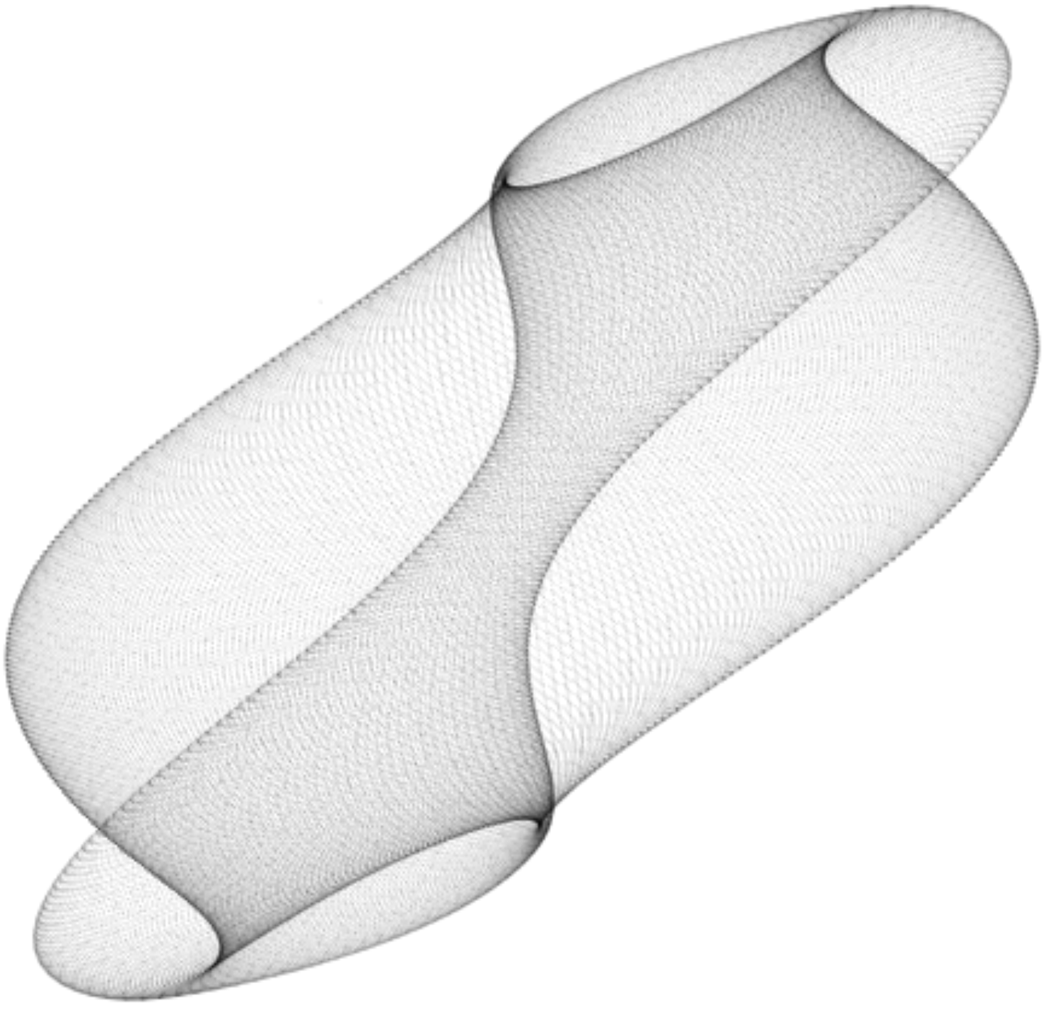
There's a lot of variables here, but what figure would we draw?

What would these points look like in 2d?







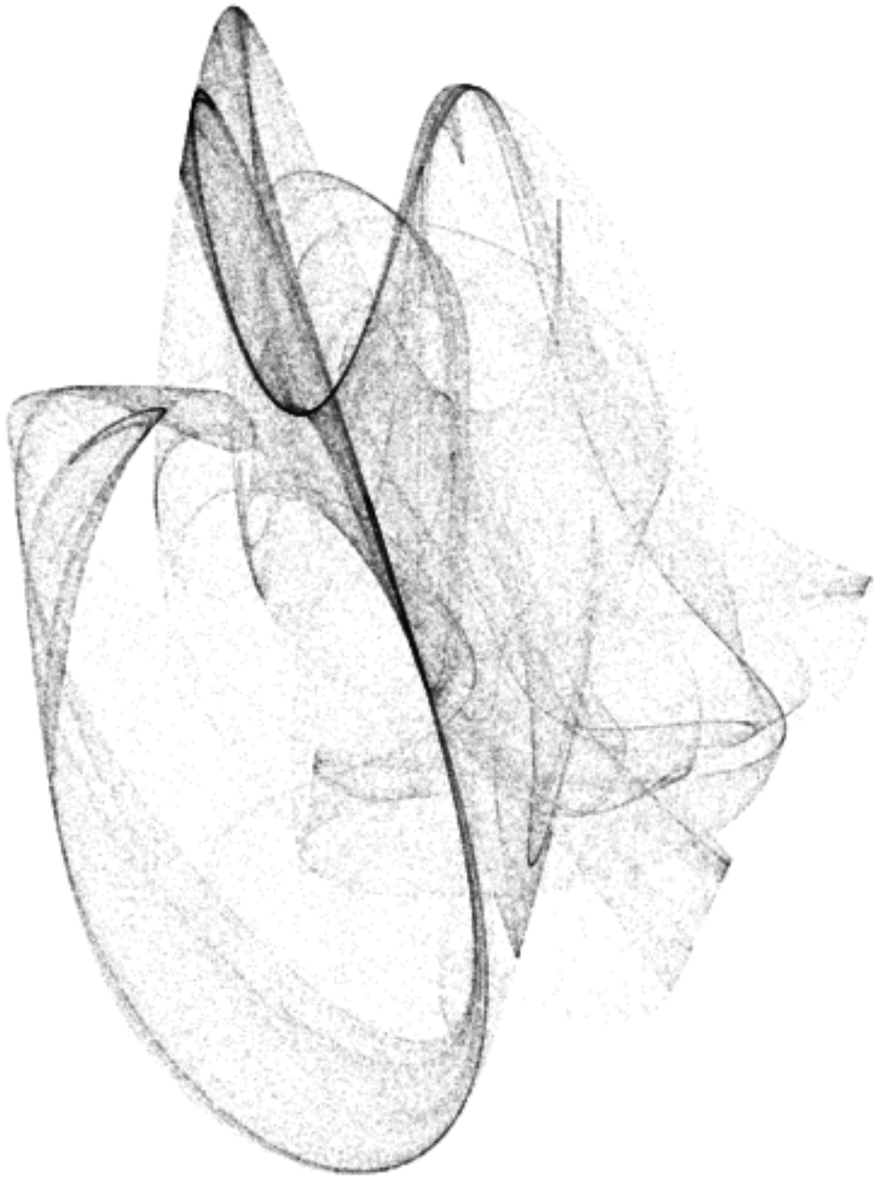


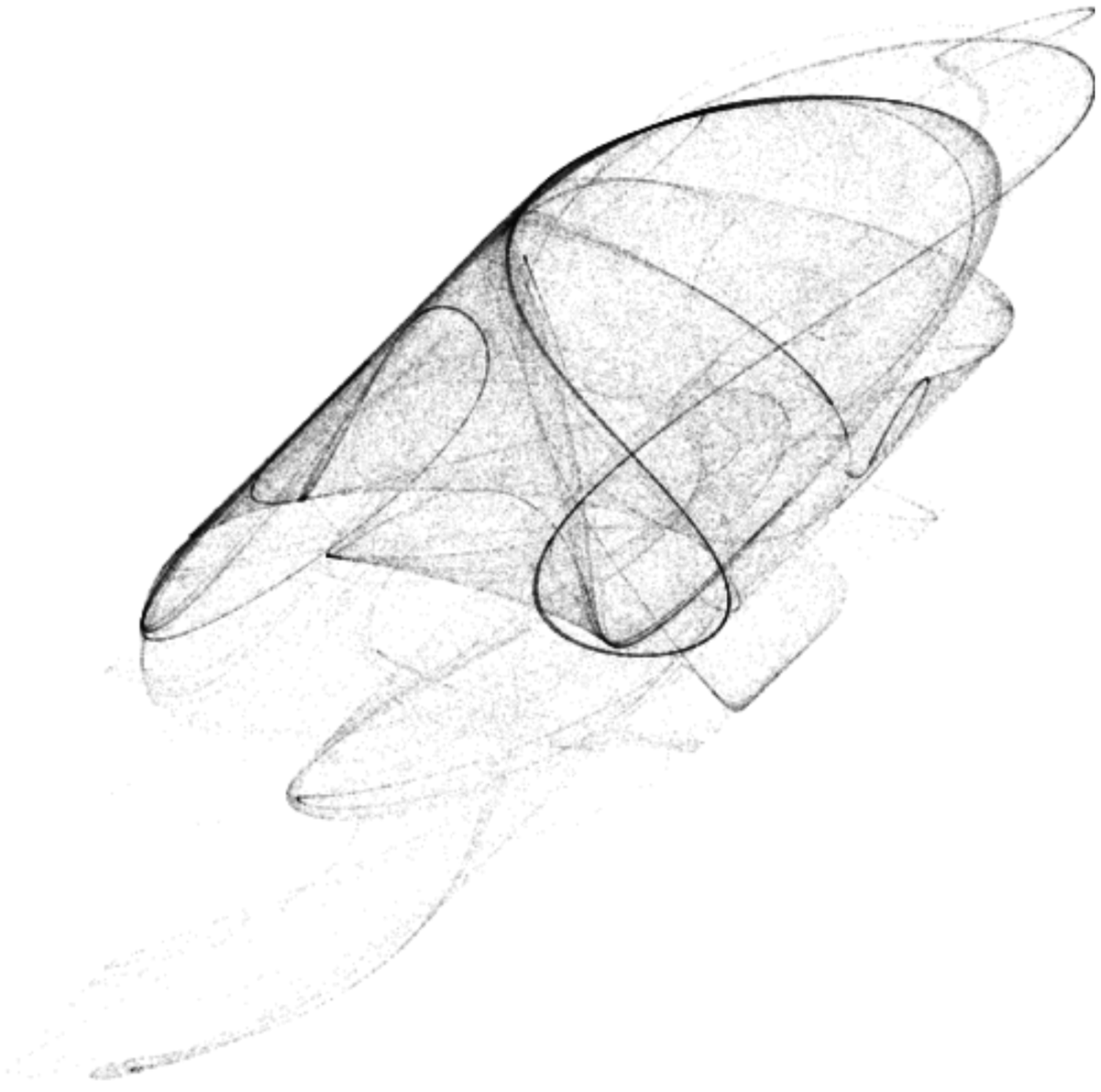
Alternative Recursion

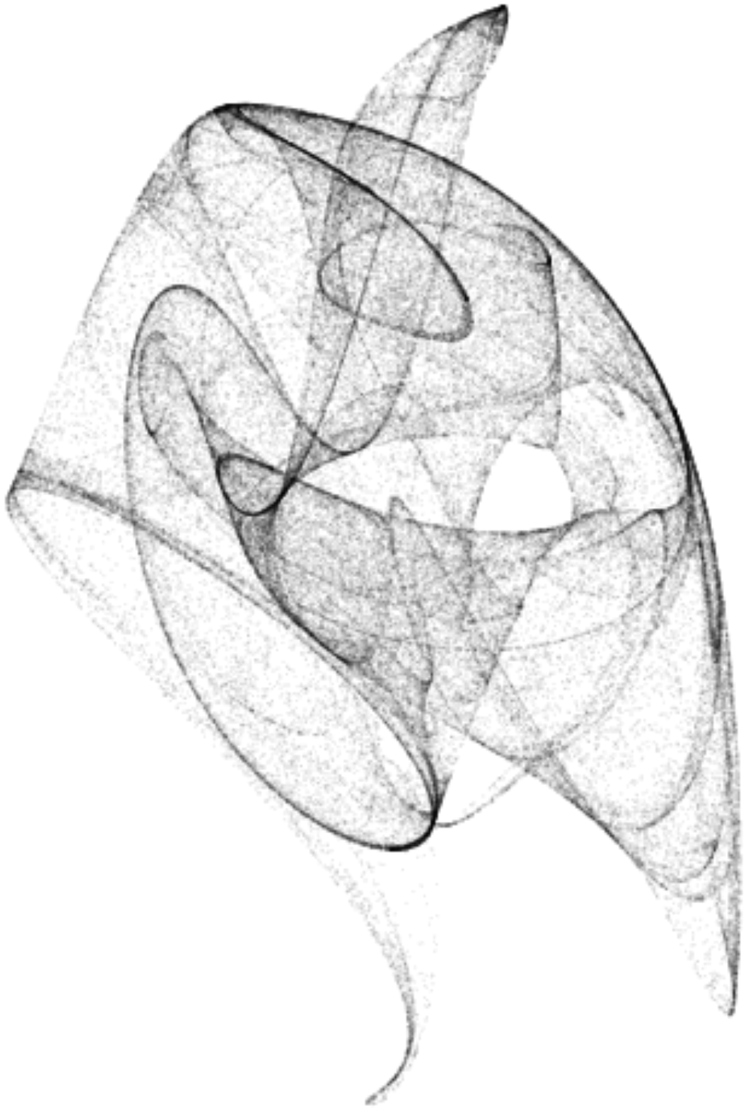
Let's change the formulas a bit.

$$x_{n+1} = \sin(a \times y_n) + \cos(b \times x_n) - \cos(c \times z_n)$$
$$y_{n+1} = \sin(d \times x_n) + \cos(e \times y_n) - \cos(f \times z_n)$$
$$z_{n+1} = x_n y_n$$

I've only changed z_{n+1}.





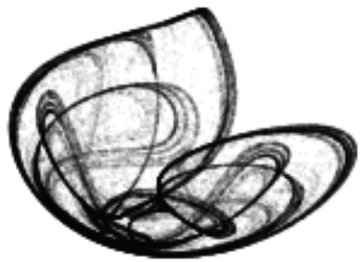


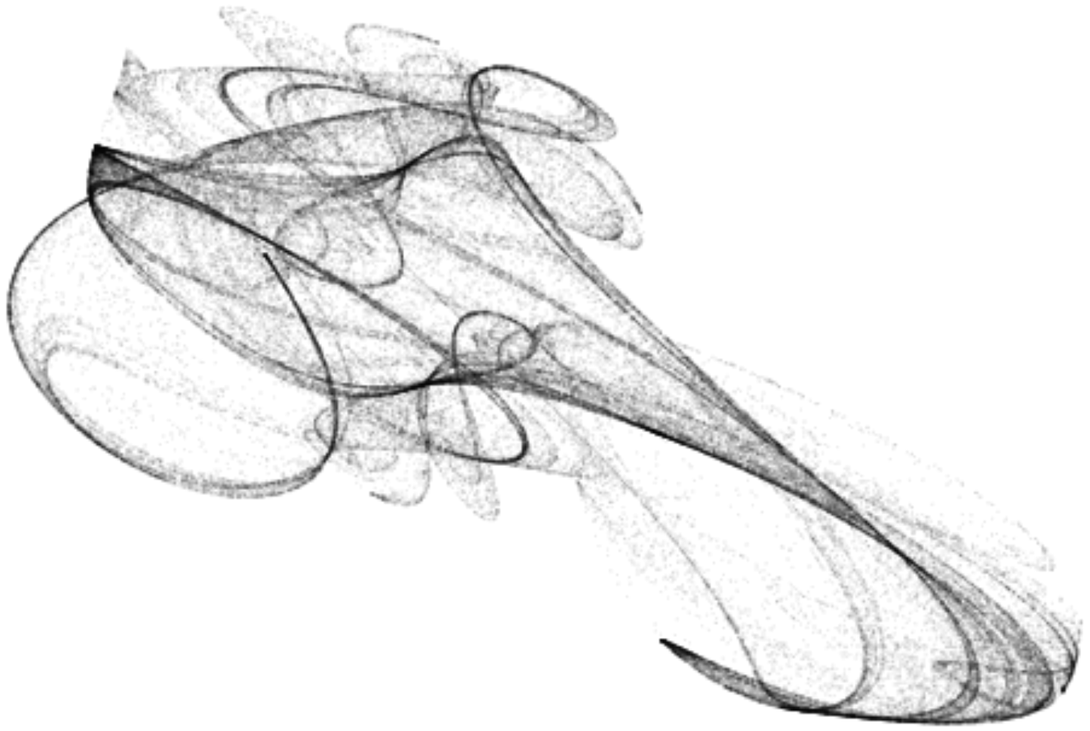
Alternative Recursion

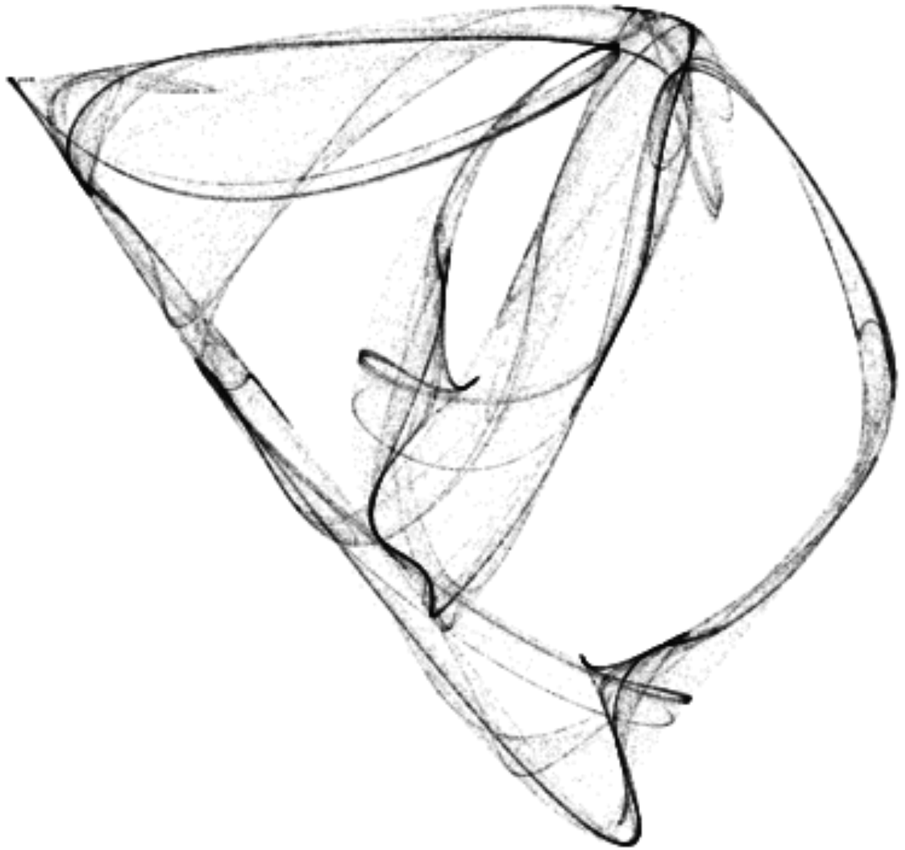
Let's change the formulas a bit.

$$x_{n+1} = \sin(a \times y_n) + \cos(b \times x_n) - \cos(c \times z_n)$$
$$y_{n+1} = \sin(d \times x_n) + \cos(e \times y_n) - \cos(f \times z_n)$$
$$z_{n+1} = \arctan(x_n + y_n)$$

I've only changed z_{n+1}.





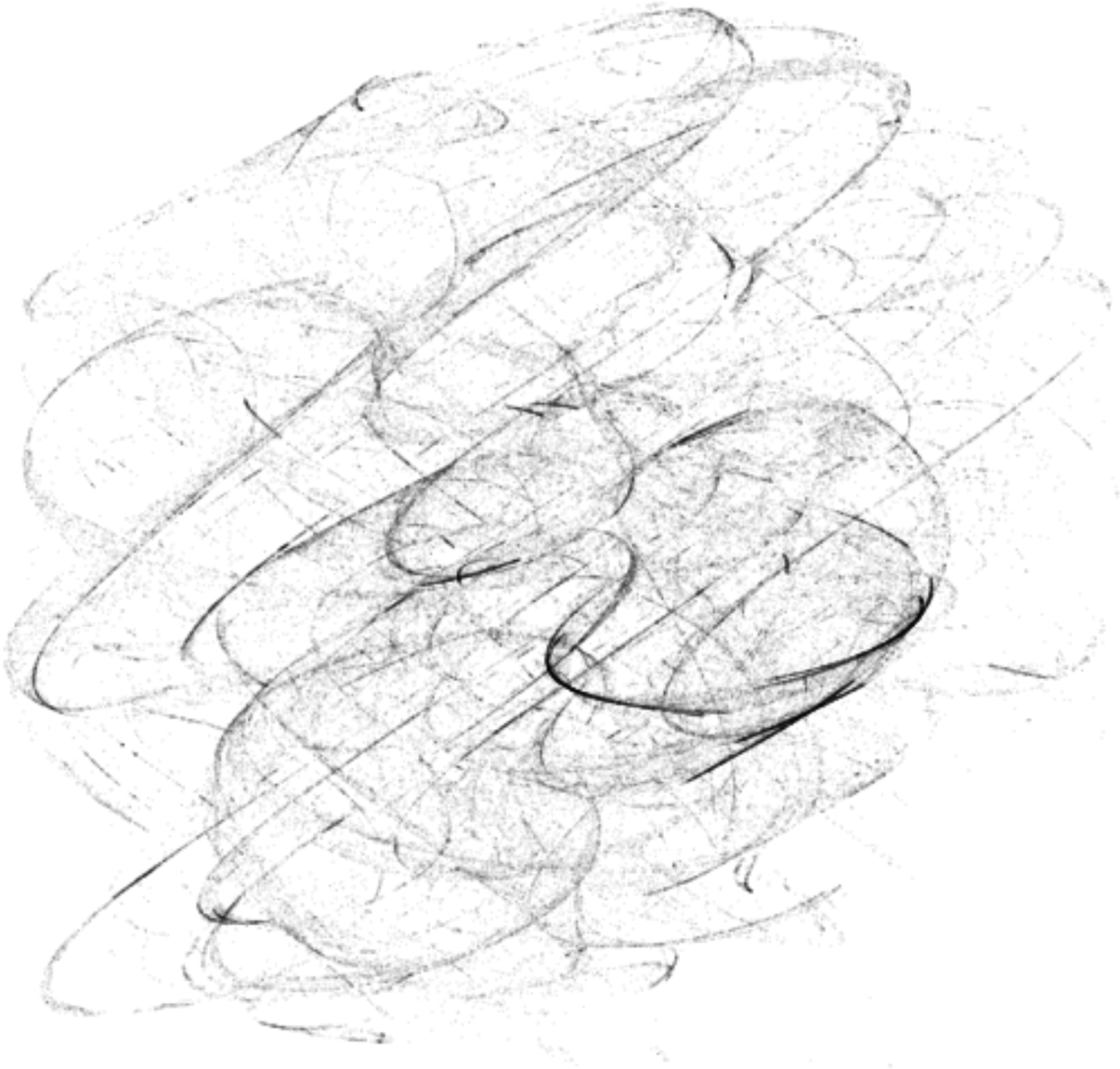


Alternative Recursion

Let's change the formulas a bit.

$$x_{n+1} = \sin(a \times y_n) + \cos(b \times x_n) - \cos(c \times z_n)$$
$$y_{n+1} = \sin(d \times x_n) + \cos(e \times y_n) - \cos(f \times z_n)$$
$$z_{n+1} = \frac{x_n^2}{x_n + y_n}$$

I've only changed z_{n+1}.



Alternative Recursion

Different formulas cause different styles of "art".

Certain values of a, b, c, d, e, f will converge to something pretty while others can feel a bit random.

We are able to draw pretty images with a very basic building block: the pixel.

Can we build this in 3d as well?

Proposal

How about we build some system like;

$$
\begin{aligned}
x_{n+1} &= \sin(a \times y_n) + \cos(b \times z_n) + \cos(c \times t_n) \\
y_{n+1} &= \sin(d \times x_n) + \cos(e \times z_n) + \cos(f \times t_n) \\
z_{n+1} &= \sin(g \times x_n) + \cos(h \times y_n) + \cos(i \times t_n) \\
t_{n+1} &= t_n + 0.1
\end{aligned}
$$

Variations are also possible.

Proposal

Also, this is 3d! We can also draw other dimensions.

$$
\begin{aligned}
x_{n+1} &= \sin(a \times y_n) + \cos(b \times z_n) + \cos(c \times t_n) \\
y_{n+1} &= \sin(d \times x_n) + \cos(e \times z_n) + \cos(f \times t_n) \\
z_{n+1} &= \sin(g \times x_n) + \cos(h \times y_n) + \cos(i \times t_n) \\
t_{n+1} &= t_n + 0.1 \\
s_{n+1} &= (\sin(t_{n+1}) + 1) \times j
\end{aligned}
$$

Assignment: Build This!

Sharing work: sketchfab

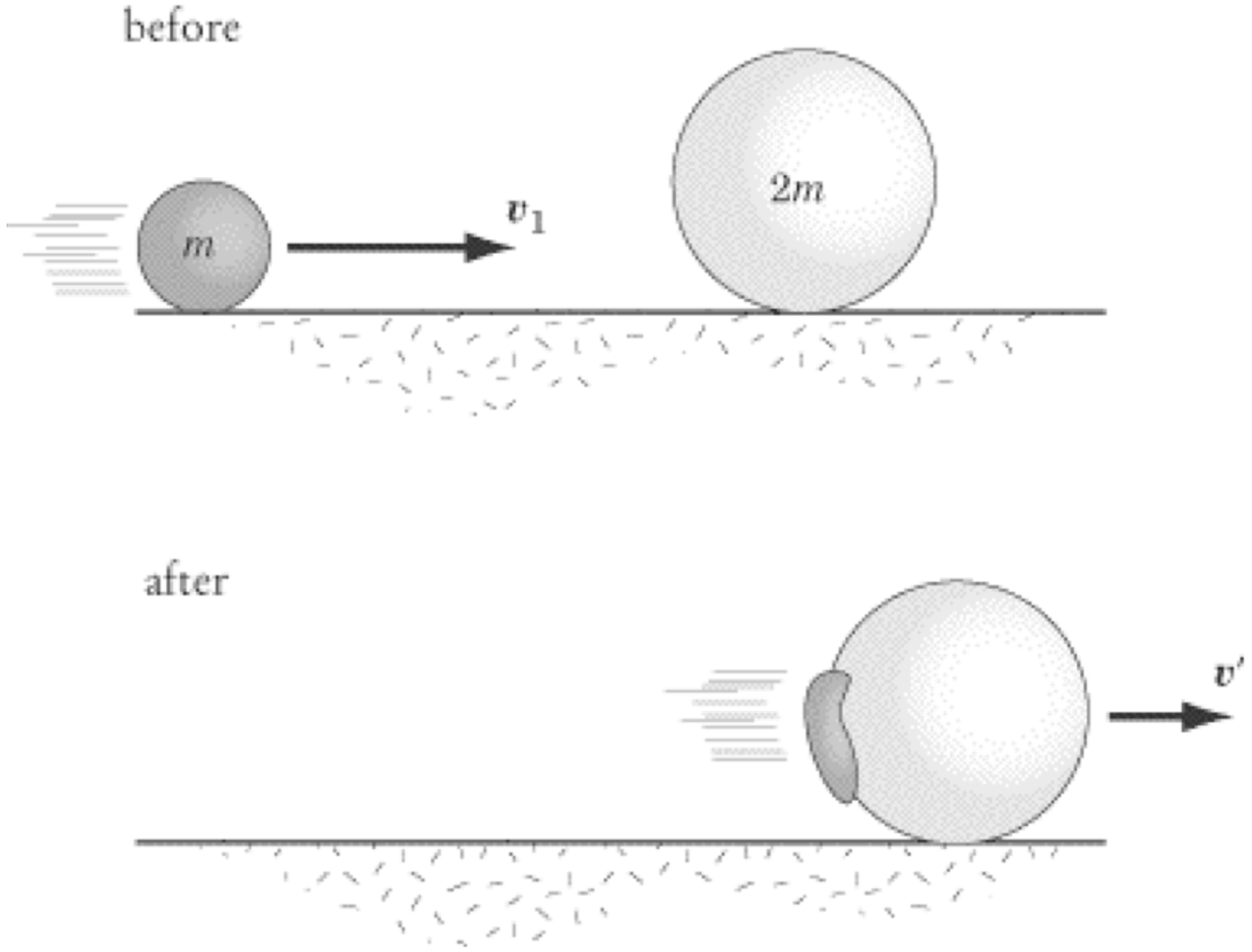
Sketchfab

Animation

Blender can render animations for us. So let's create an animation.

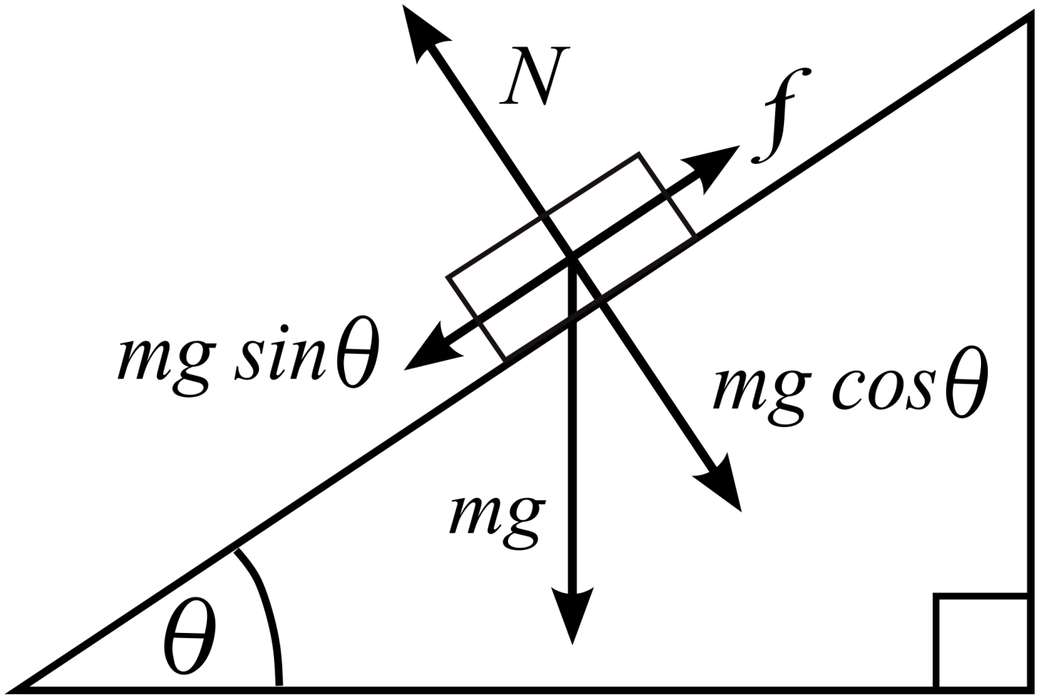
Let's first see a cool example of what you can do here and here.

Physics



Animation

Blender has a physics engine so how about we do something with falling objects/explosions?



Assigning Physics

You'll need to specify a falling object to be an "activate rigid body".

```
bpy.ops.mesh.primitive_cube_add(radius=1, location = (0,0,10))
bpy.ops.rigidbody.object_add()
bpy.context.active_object.rigid_body.type = 'ACTIVE'
```

You'll need to specify a static object to be an "passive rigid body".

```
bpy.ops.mesh.primitive_plane_add(radius = 10, location = (0,0,0))
bpy.ops.rigidbody.object_add()
bpy.context.active_object.rigid_body.type = 'PASSIVE'
```

Experiment

Generate many cubes on the same place and see what happens.

```
bpy.ops.mesh.primitive_cube_add(radius=1, location = (0,0,10))
bpy.ops.rigidbody.object_add()
bpy.context.active_object.rigid_body.type = 'ACTIVE'
```

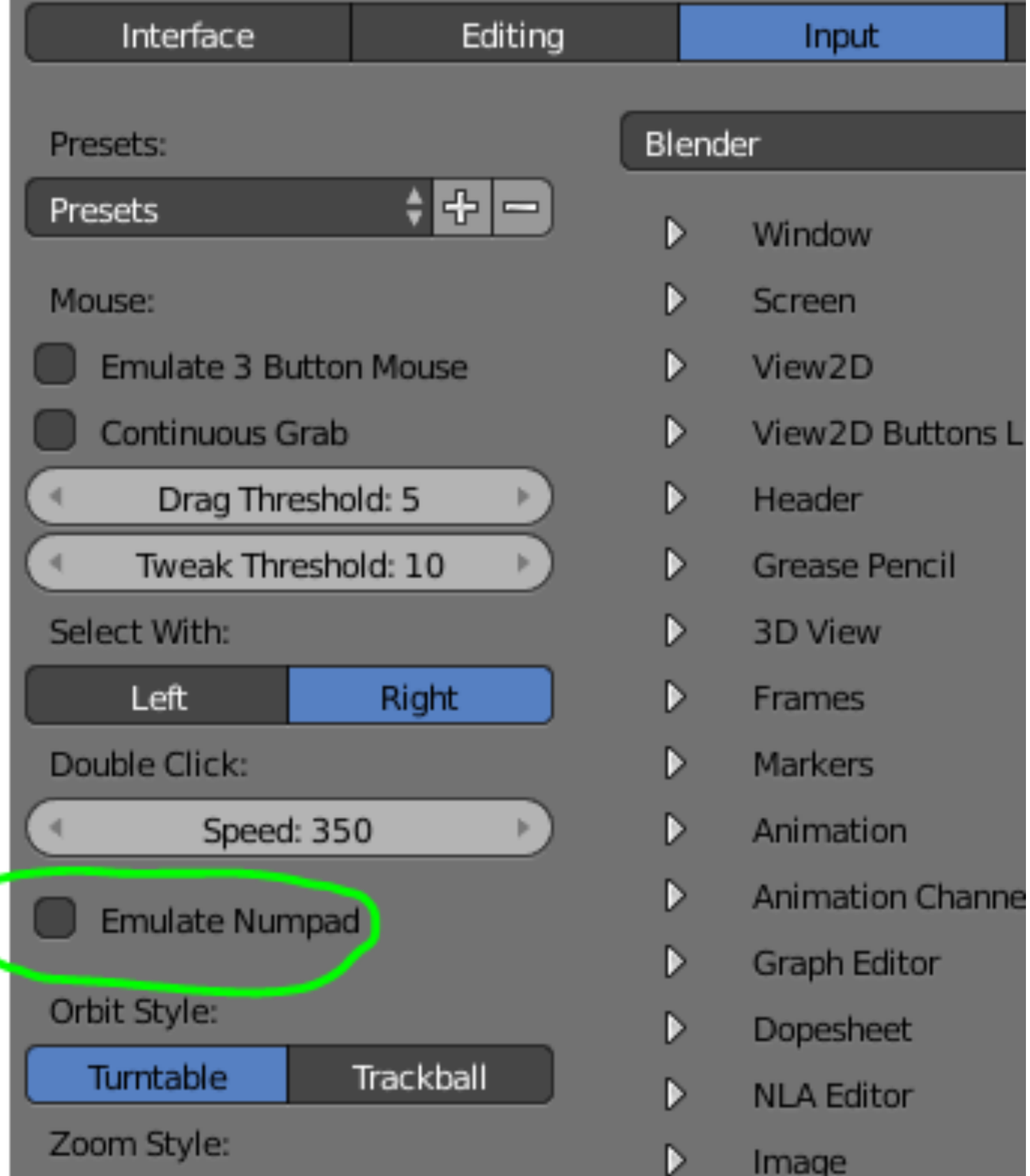
Why does this happen?

Falling Tower/Exploding Cubes

Vincent will now copy and paste code into codeshare from his blogpost. He will walk you through the code: you can play with it.

The idea is to construct a large tower of many small cubes that will fall on a plateau.

If the computer crashes: draw less cubes!



Protip

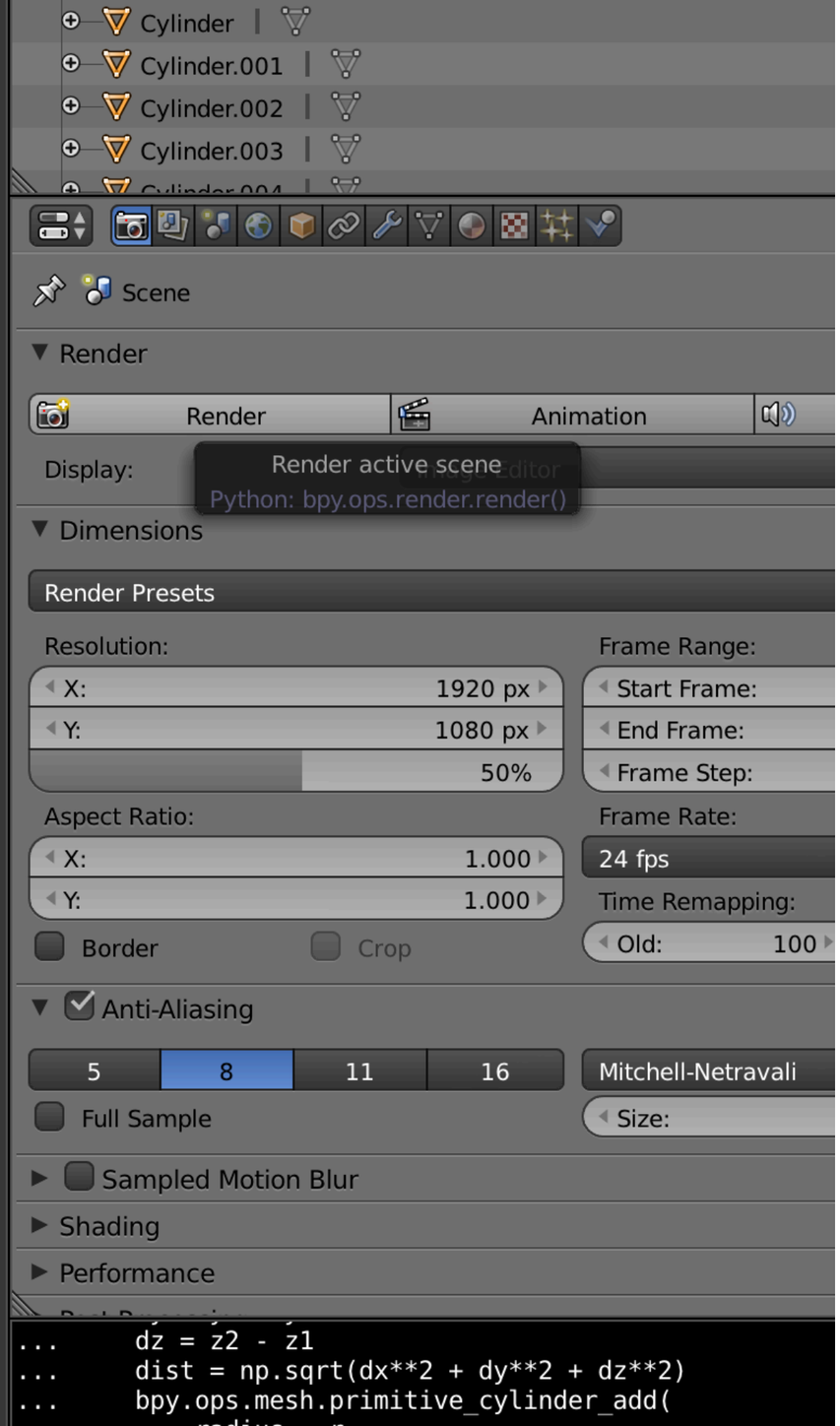
Go to user Settings.

Select "input".

Click "Emulate Numpad"

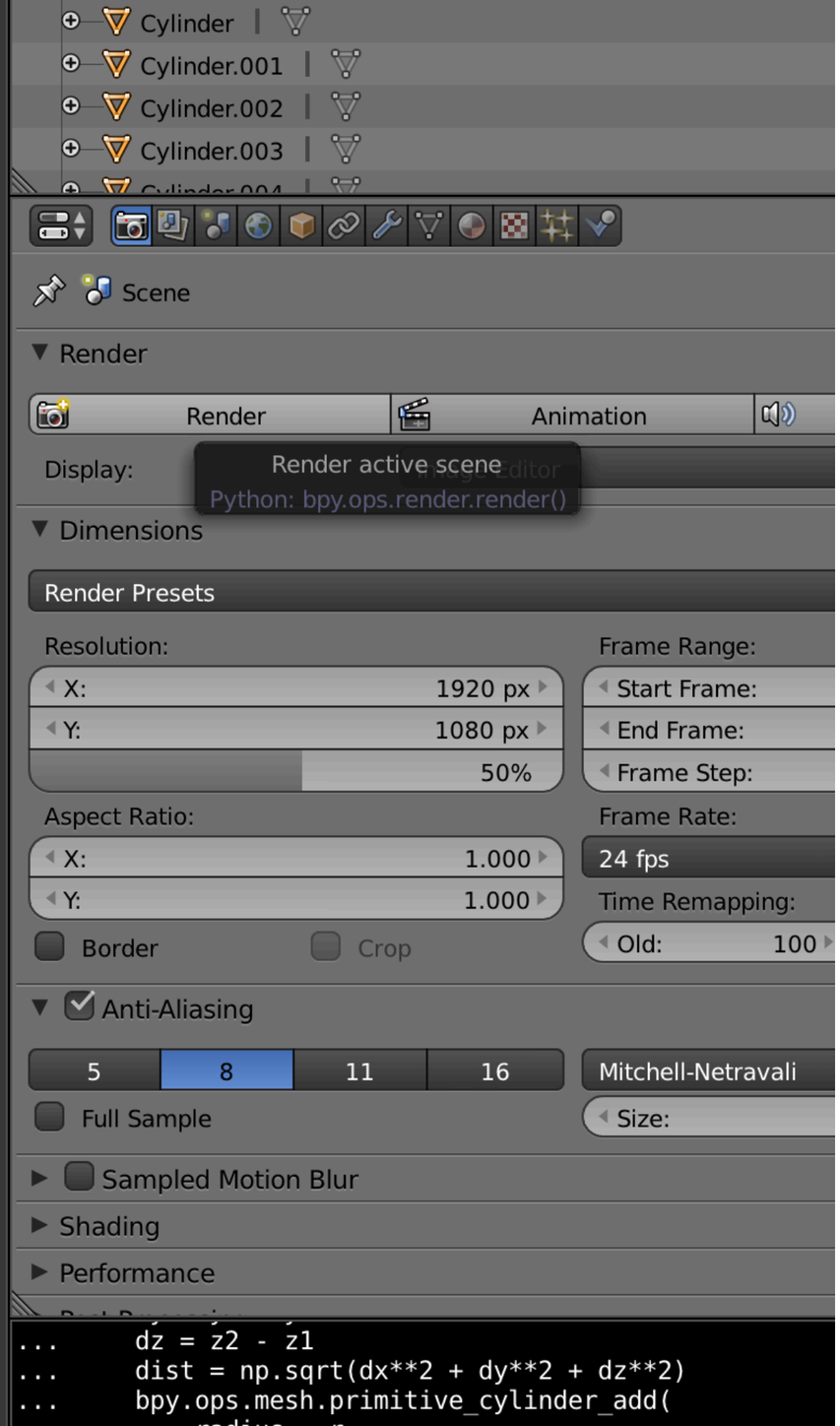
You'll then have an awesome shortcut available to you.

You can move the camera to your viewpoint with: **CTRL+ALT+0**.



Rendering

Once you have a nice animation, you'll probably want to render it. Make sure you've got a lightsource and a camera then press the render button.



Rendering

You'll have the option of generating images or to generate a video. These will be usually be placed in `/tmp/`.

Rendering

Alternatively, you can use this code.

```
# these are the render settings
bpy.data.scenes['Scene'].render.engine = 'CYCLES'
bpy.data.scenes['Scene'].cycles.samples = 10
bpy.data.scenes['Scene'].frame_end = 100
bpy.data.scenes['Scene'].render.fps = 100
bpy.context.scene.render.resolution_x = 600
bpy.context.scene.render.resolution_y = 400

# this command signals the actual render
bpy.ops.render.render(animation=True, use_viewport=True)
```

Thanks

Here's some links for those interested

- my blog, blogpost 1
- my blog, blogpost 2
- pretty good tutorial
- sketchfab
- blender main website
- blender documentation