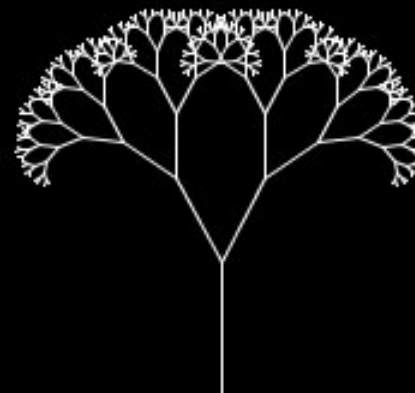
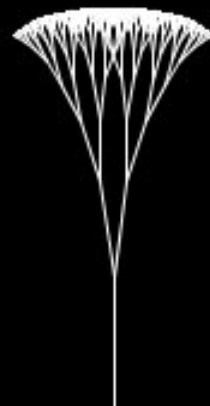


DIGM 360: Computer Game Development:

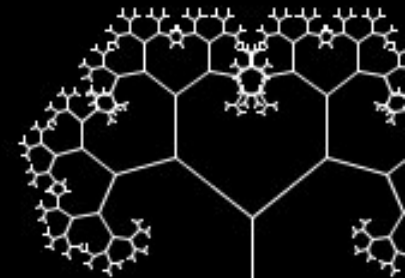
A Brief Introduction to 3D Programming for the Game Environment



Will Muto
Digital Media
Drexel University

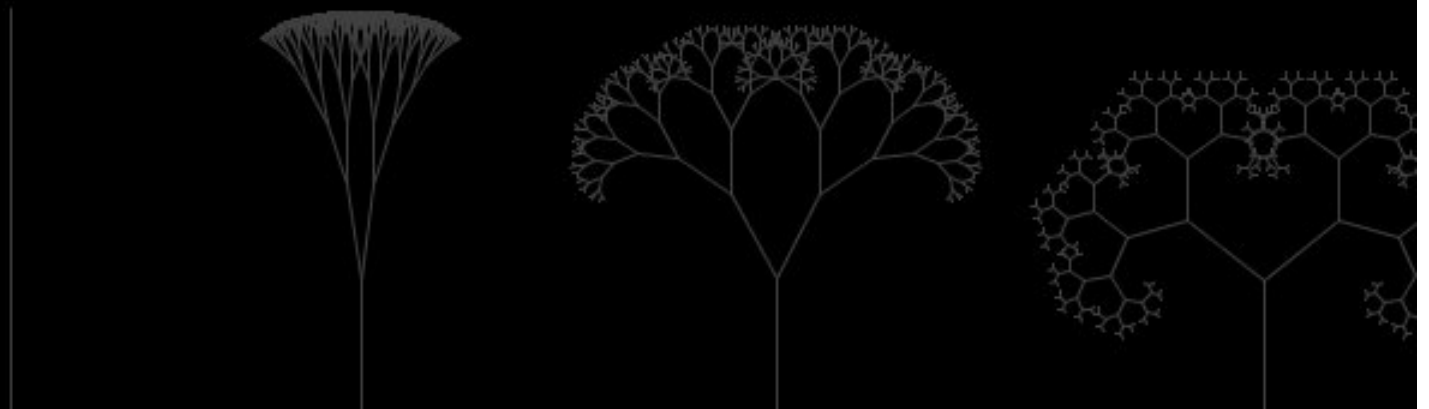


Recursive Tree by Daniel Shiffman



A quick review

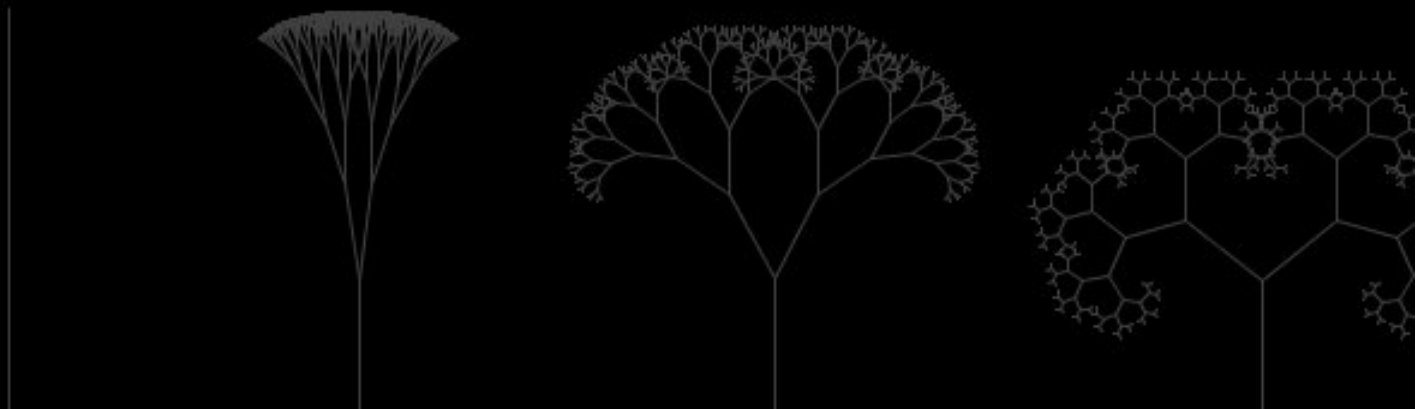
OOP (Object Oriented Programming)
& Inheritance
Buffers



A quick review: OOP & Inheritance



Object Car



A quick review: OOP & Inheritance



Object Car

Properties:

Color

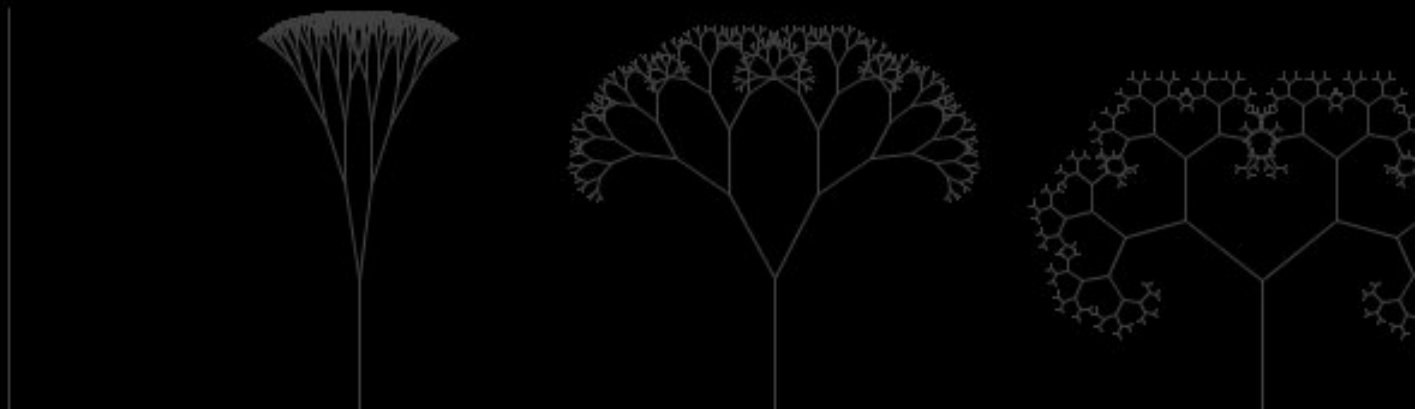
Wheels

Methods:

Forward

Reverse

Change Gear



A quick review: OOP & Inheritance



Object Car

Properties:

Color

Wheels

Methods:

Forward

Reverse

Change Gear



Object BMW inherits Car

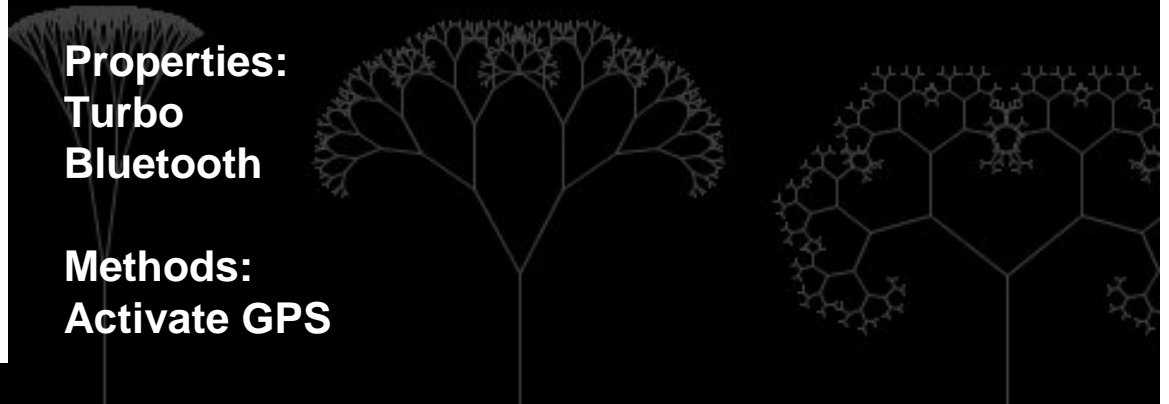
Properties:

Turbo

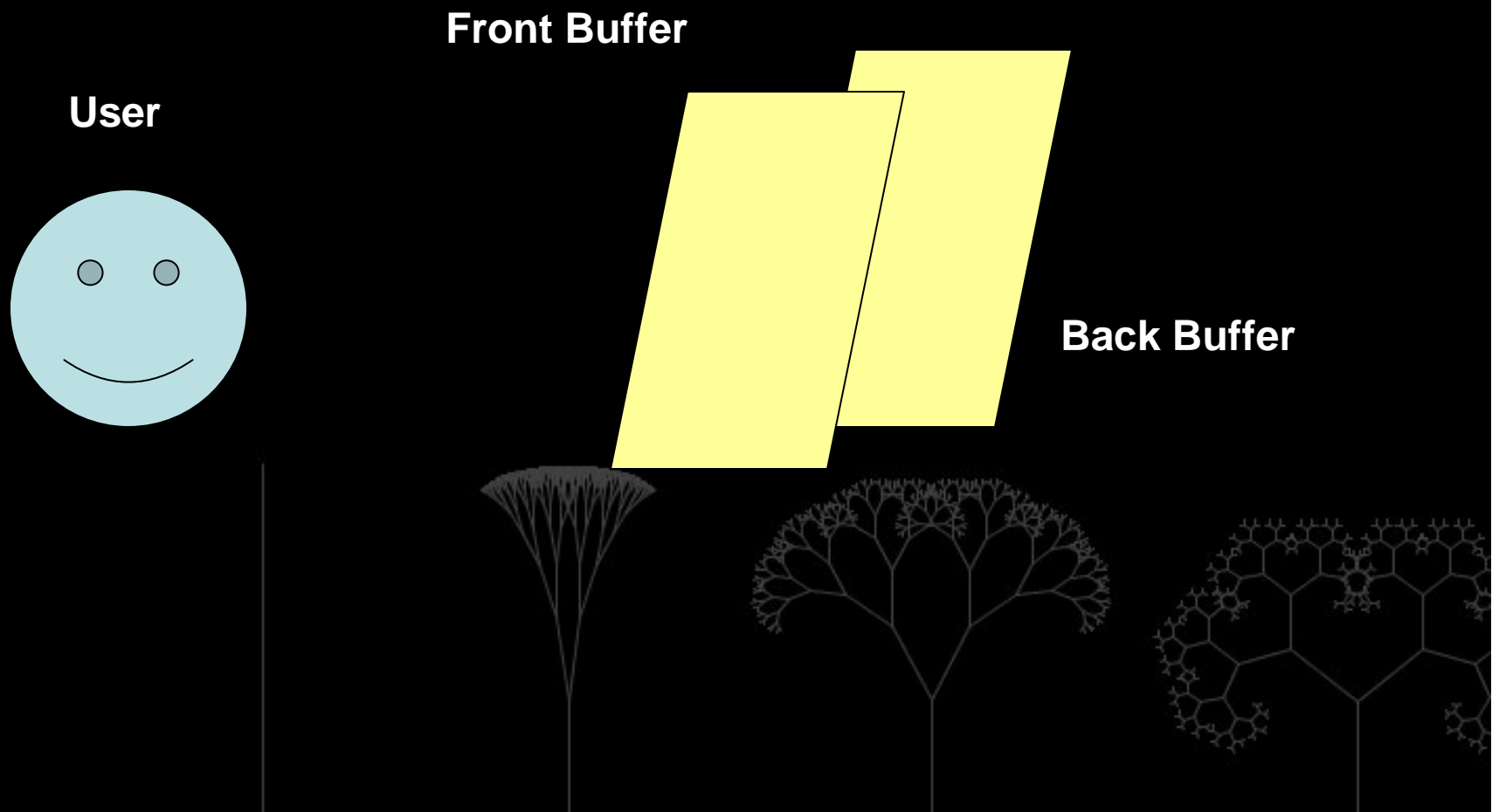
Bluetooth

Methods:

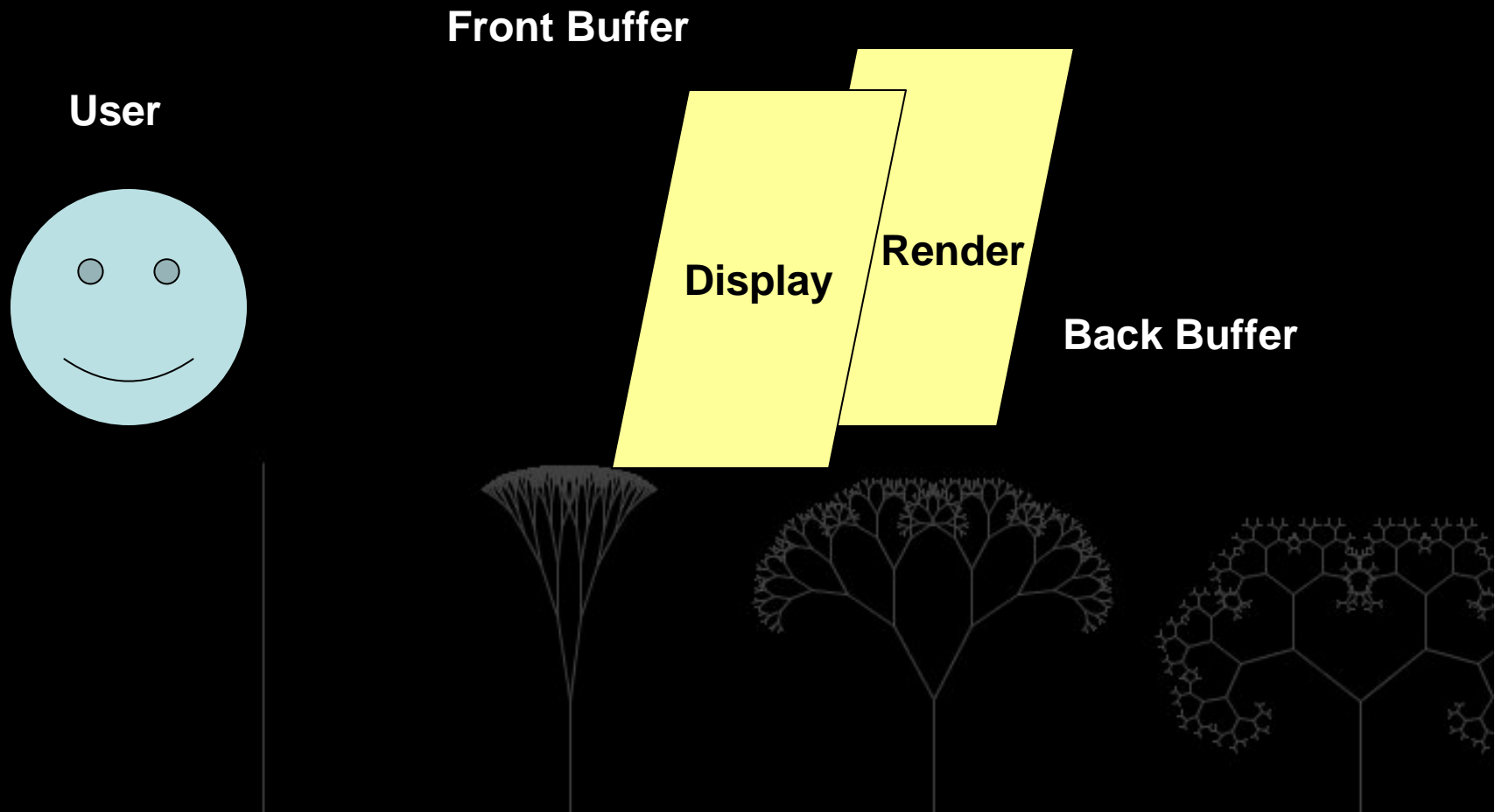
Activate GPS



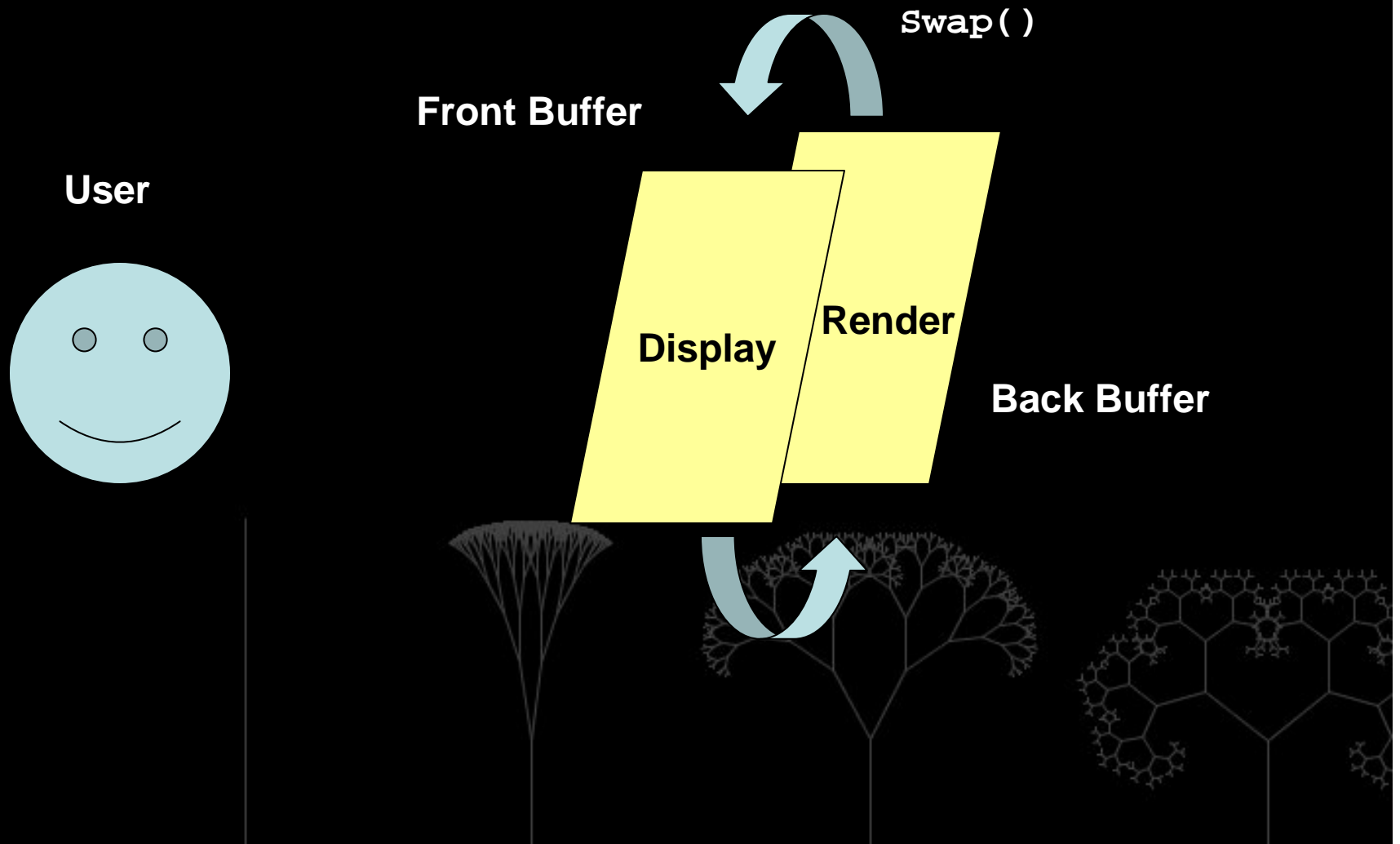
A quick review: Buffers



A quick review: Buffers

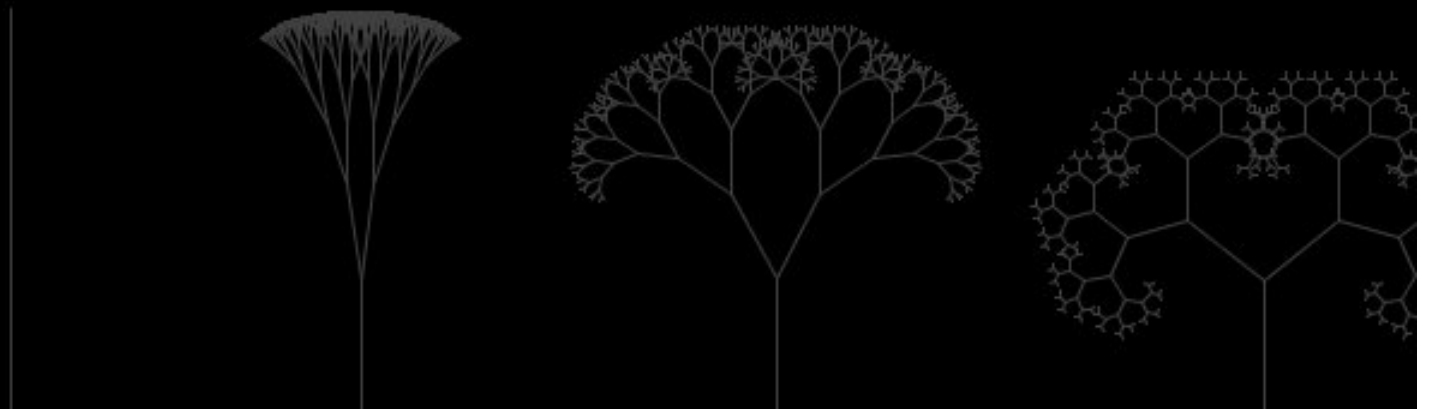


A quick review: Buffers



Wait a gosh-darned minute...

I am an artist, why the heck should
I care about programming?



“The ability to ‘read’ a medium means you can access materials and tools created by others. The ability to ‘write’ in a medium means you can generate materials and tools for others. You must have both to be literate.”

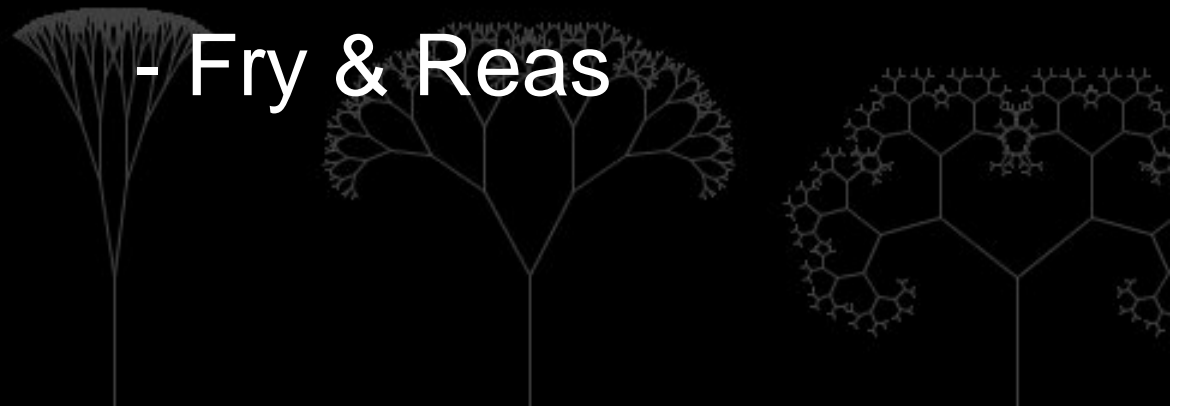
- Alan Kay, Xerox PARC & Apple



“Programming is not just for engineers”

“To fully explore the computer as an artistic material, its important to understand this ‘arcane art of computer programming.’”

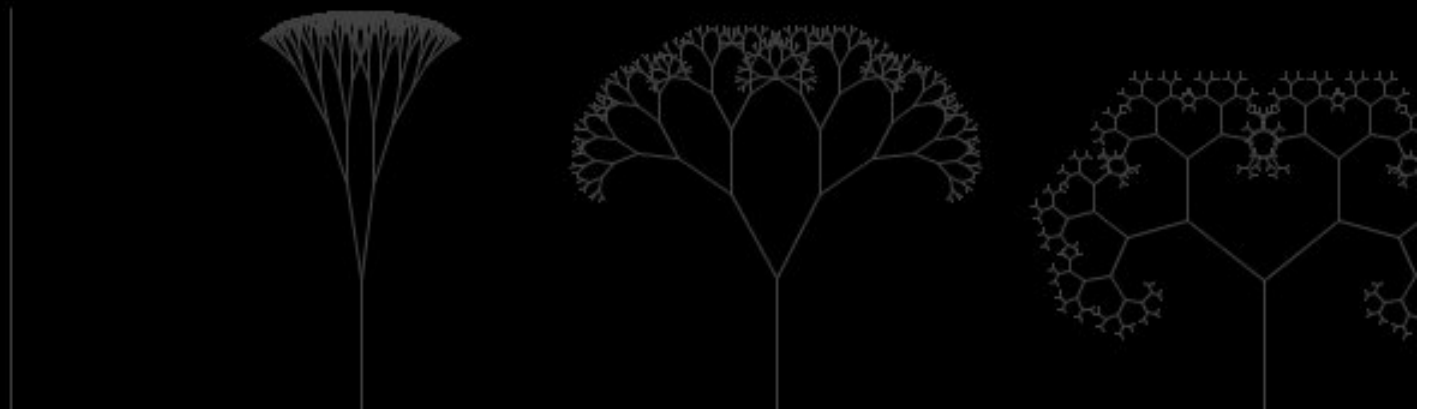
- Fry & Reas



Keepin' It Simple with Processing

“Processing relates software concepts to principles of visual form, motion, and interaction.”

- Fry & Reas




Keepin' It Simple with Processing

- Java-based, free programming language
- Simplified style and syntax aimed at artists with little programming experience
- Provides access to advanced features for experts
- Started by Ben Fry and Casey Reas and evolved in the MIT Media Lab
- Can be used to interface with hardware (Arduindo) as well as mobile devices
- Not necessarily a game language, but can be utilized as such

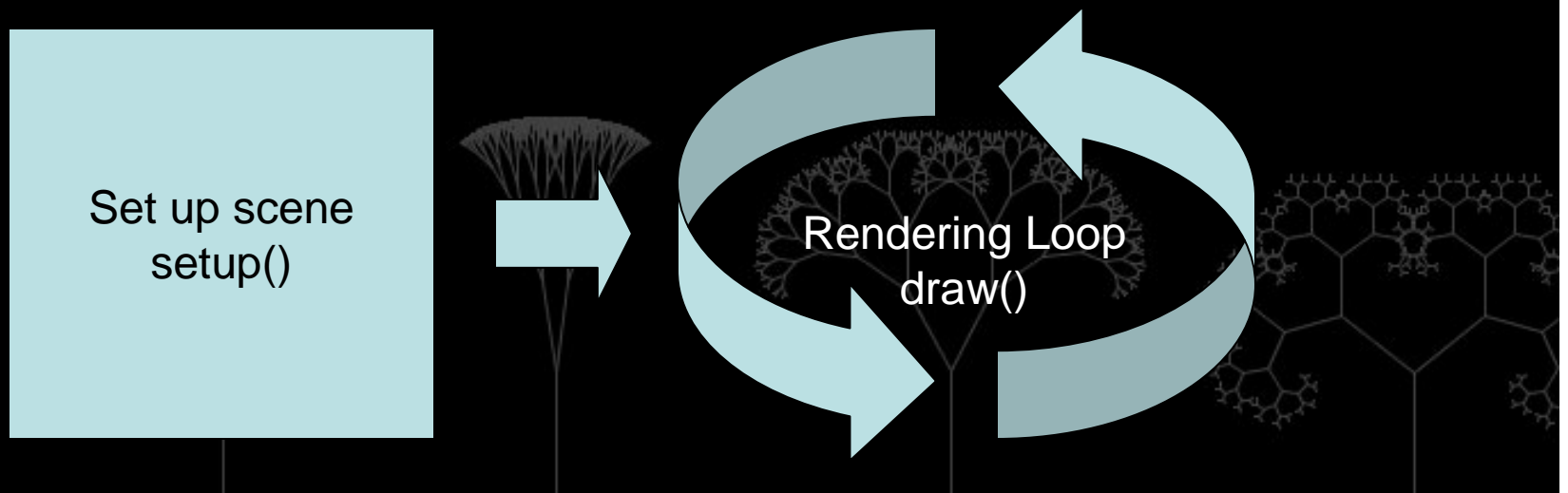


Keepin' It Simple with Processing

- Parses input from keyboard, joysticks
 - Network support
 - Computer vision libraries
 - OpenGL rendering
 - GUI
 - And, since it's built on Java, **anything else Java supports** natively or through APIs
- 

Keepin' It Simple with Processing

- Even at its simplest, works in the same way many large-scale engines do
- Doesn't require users to understand advanced concepts like double-buffering, but still makes them available



Keepin' It Simple with Processing

- Simple IDE
 - Debugging
 - Auto Format
 - Compiler
 - Examples



```
Processing - 0125 Beta
File Edit Sketch Tools Help

TexturedSphere

/**
 * Textured Sphere
 * by Mike 'Flux' Chang (cleaned up by Aaron Koblin).
 * Based on code by Toxi.
 *
 * A 3D textured sphere with simple rotation control.
 * Note: Controls will be inverted when sphere is upside down.
 * Use an "arc ball" to deal with this appropriately.
 */

import processing.opengl.*;

PImage bg;
PImage texmap;

int sDetail = 35; //Sphere detail setting
float rotationX = 0;
float rotationY = 0;
float velocityX = 0;
float velocityY = 0;
float globeRadius = 300;
float pushBack = 0;

float[] cx,cz,sphereX,sphereY,sphereZ;
float sinLUT[];
float cosLUT[];
float SINCOS_PRECISION = 0.5f;
int SINCOS_LENGTH = int(360.0 / SINCOS_PRECISION);

void setup()
{
  1
```

Keepin' It Simple with Processing

- Simple IDE
 - Debugging
 - Auto Format
 - Compiler
 - Examples



```
Processing - 0125 Beta
File Edit Sketch Tools Help
[Play] [Save] [Copy] [Paste] [Undo] [Redo]
TexturedSphere
/**
 * Textured Sphere
 * by Mike 'Flux' Chang (cleaned up by Aaron Koblin).
 * Based on code by Toxi.
 *
 * A 3D textured sphere with simple rotation control.
 * Note: Controls will be inverted when sphere is upside down.
 * Use an "arc ball" to deal with this appropriately.
 */

import processing.opengl.*;

PImage bg;
PImage texmap;

int sDetail = 35; //Sphere detail setting
float rotationX = 0;
float rotationY = 0;
float velocityX = 0;
float velocityY = 0;
float globeRadius = 300;
float pushBack = 0;

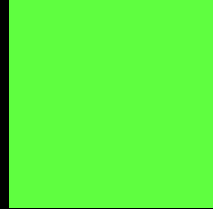
float[] cx,cz,sphereX,sphereY,sphereZ;
float sinLUT[];
float cosLUT[];
float SINCOS_PRECISION = 0.5f;
int SINCOS_LENGTH = int(360.0 / SINCOS_PRECISION);

void setup()
{
  1
```

Keepin' It Simple with Processing

- Creating color
 - RGB/HSV
 - Triplet or Hex notation

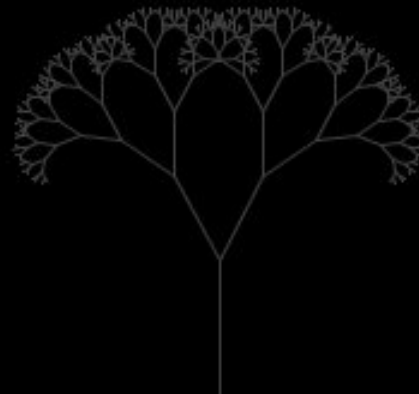
```
colorMode(RGB);  
color(95, 255, 64) =
```



```
colorMode(RGB);  
color(#5fff40) =
```



```
colorMode(HSV);  
color(349, 93, 84) =
```

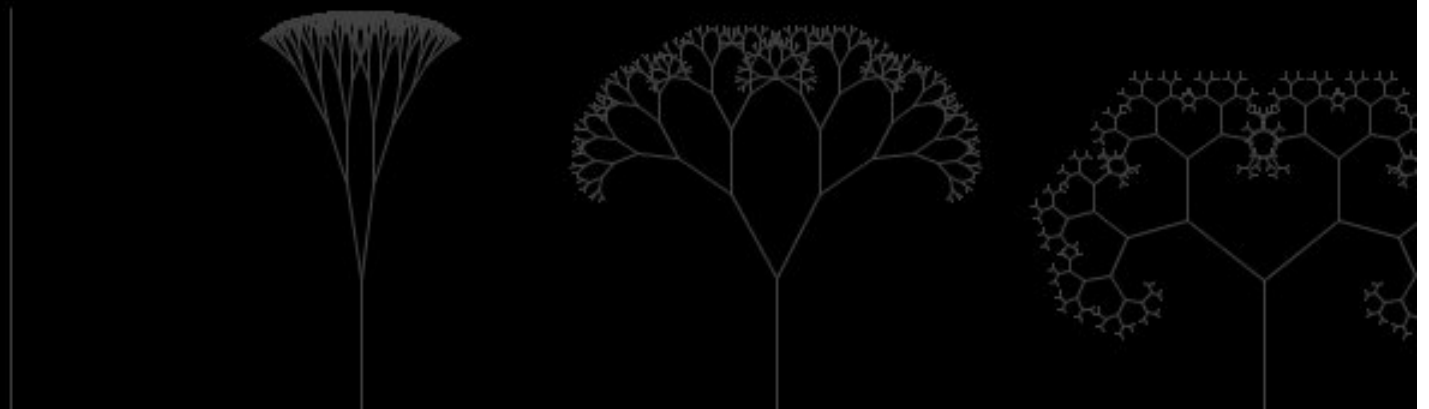


Keepin' It Simple with Processing

- Creating forms

```
point(x,y);  
line(x1,y1,x2,y2);  
triangle(x1,y1,x2,y2,x3,y3);  
quad(x1,y1,x2,y2,x3,y3,x4,y4);
```

```
box(width, height, depth);  
sphere(radius);
```

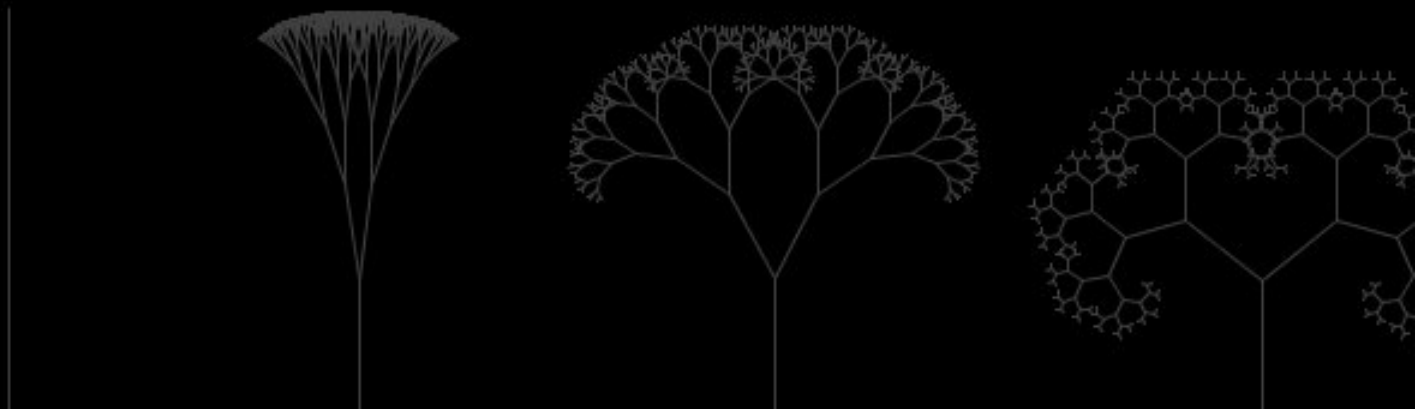


Keepin' It Simple with Processing

- Capturing input

```
mouseX  
mouseY  
mousePressed()  
mouseReleased()
```

```
key  
keyPressed()
```



Keepin' It Simple with Processing

```
// hello world
// a simple, boring cube

void setup () { // called once
  size(800,600, P3D); // define window size
  colorMode(RGB);
}

void draw() { // called every frame
  background(0.5, 0.5, 0.45);
  translate(width/2, height/2, -30);
  scale(10);
  box(20);
}
```



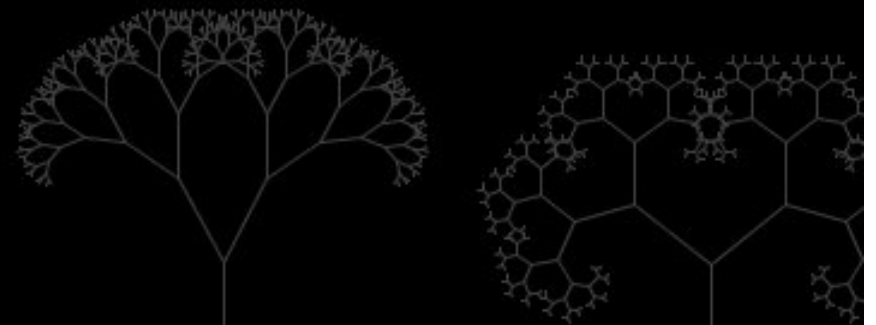
Keepin' It Simple with Processing

```
// hello world 1.1
// a slightly more interesting cube

import processing.opengl.*;
float xmag, ymag = 0;

void setup () {                                // called once
    size(800,600, OPENGL);                    // define window size
    colorMode(RGB);
}

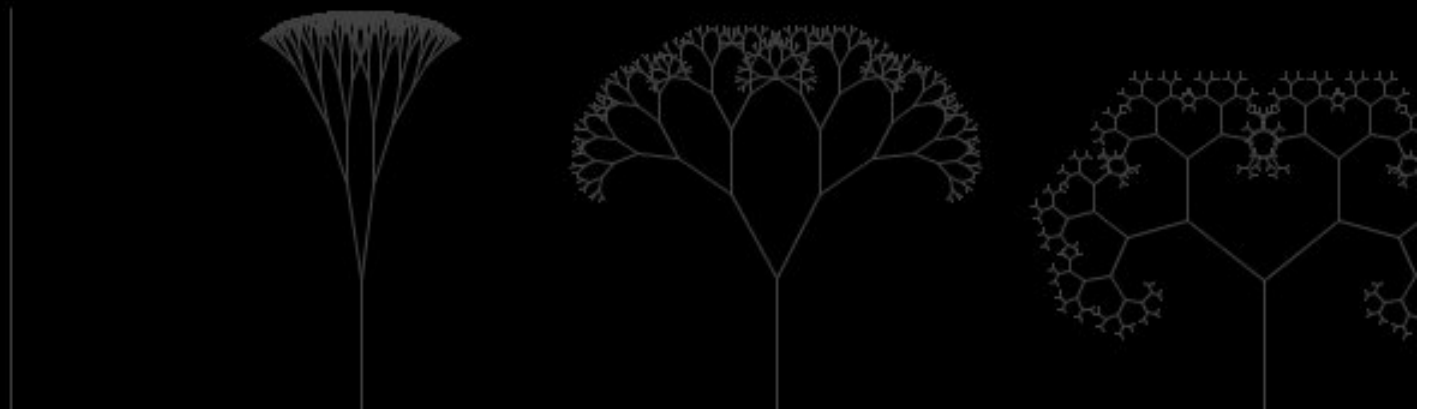
void draw() {                                  // called every frame
    background(0.5, 0.5, 0.45);
    pushMatrix();
    translate(width/2, height/2, -30);
    xmag = mouseX/float(width) * TWO_PI;
    ymag = mouseY/float(height) * TWO_PI;
    scale(20);
    rotateX(-ymag);
    rotateY(-xmag);
    box(20);
    popMatrix();
}
```



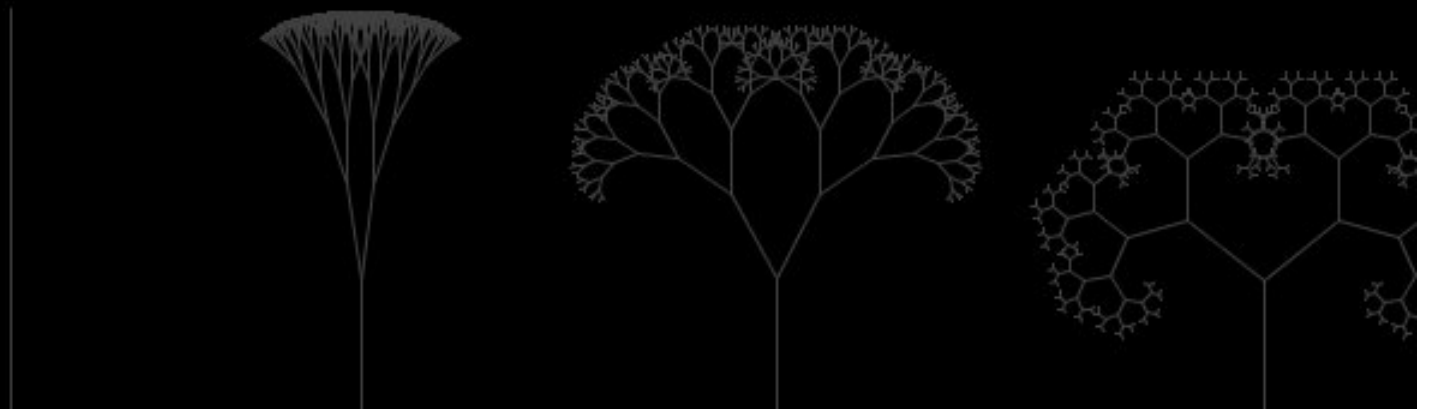
Keepin' It Simple with Processing

- Complete reference:

http://processing.org/reference/index_ext.html



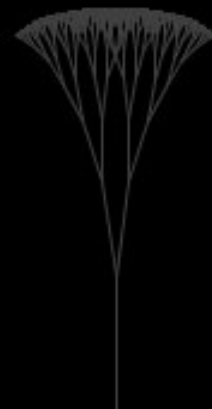
Act 2: Applying these concepts to a slightly more sophisticated gaming language



Blitz3D

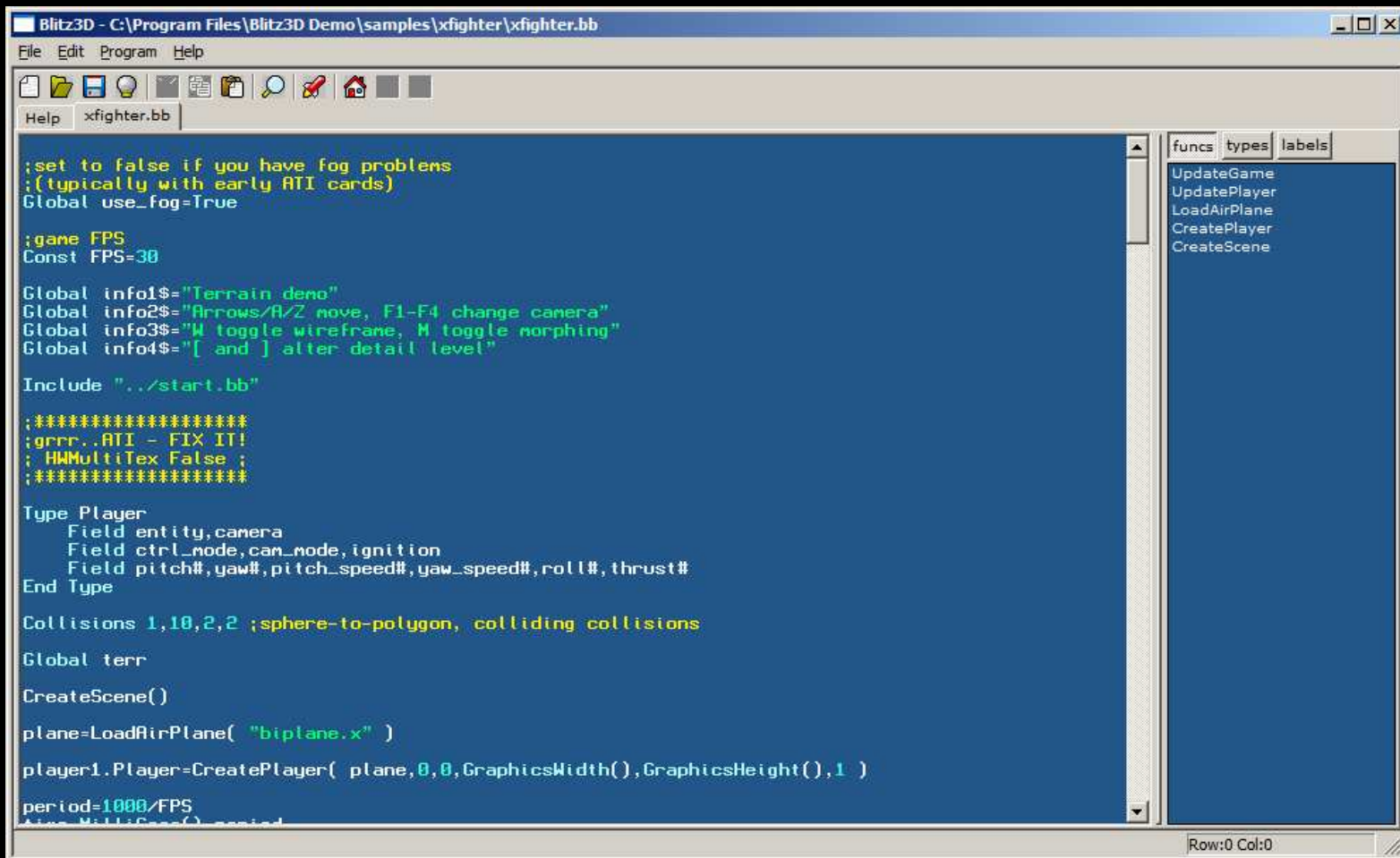
<http://www.blitzbasic.com>

- Scripting language: BlitzBasic
- Free online documentation, samples
- Plug-ins for physics, environment
- Separate GUI package
- Not particularly well-suited for large projects
- Import from: Truespace, Lightwave3D, 3DS Max, Milkshape3D



Blitz3D

- Another simple IDE



```
Blitz3D - C:\Program Files\Blitz3D Demo\samples\xfighter\xfighter.bb
File Edit Program Help

Help xfighter.bb

;set to false if you have fog problems
;(typically with early ATI cards)
Global use_fog=True

;game FPS
Const FPS=30

Global info1$="Terrain demo"
Global info2$="Arrows/A/Z move, F1-F4 change camera"
Global info3$="W toggle wireframe, M toggle morphing"
Global info4$="[ and ] alter detail level"

Include "../start.bb"

;*****
;grrr..ATI - FIX IT!
;HMMultiTex False;
;*****

Type Player
  Field entity,camera
  Field ctrl_node,can_node,ignition
  Field pitch#,yaw#,pitch_speed#,yaw_speed#,roll#,thrust#
End Type

Collisions 1,10,2,2 ;sphere-to-polygon, colliding collisions

Global terr

CreateScene()

plane=LoadAirPlane( "biplane.x" )

player1.Player=CreatePlayer( plane,0,0,GraphicsWidth(),GraphicsHeight(),1 )

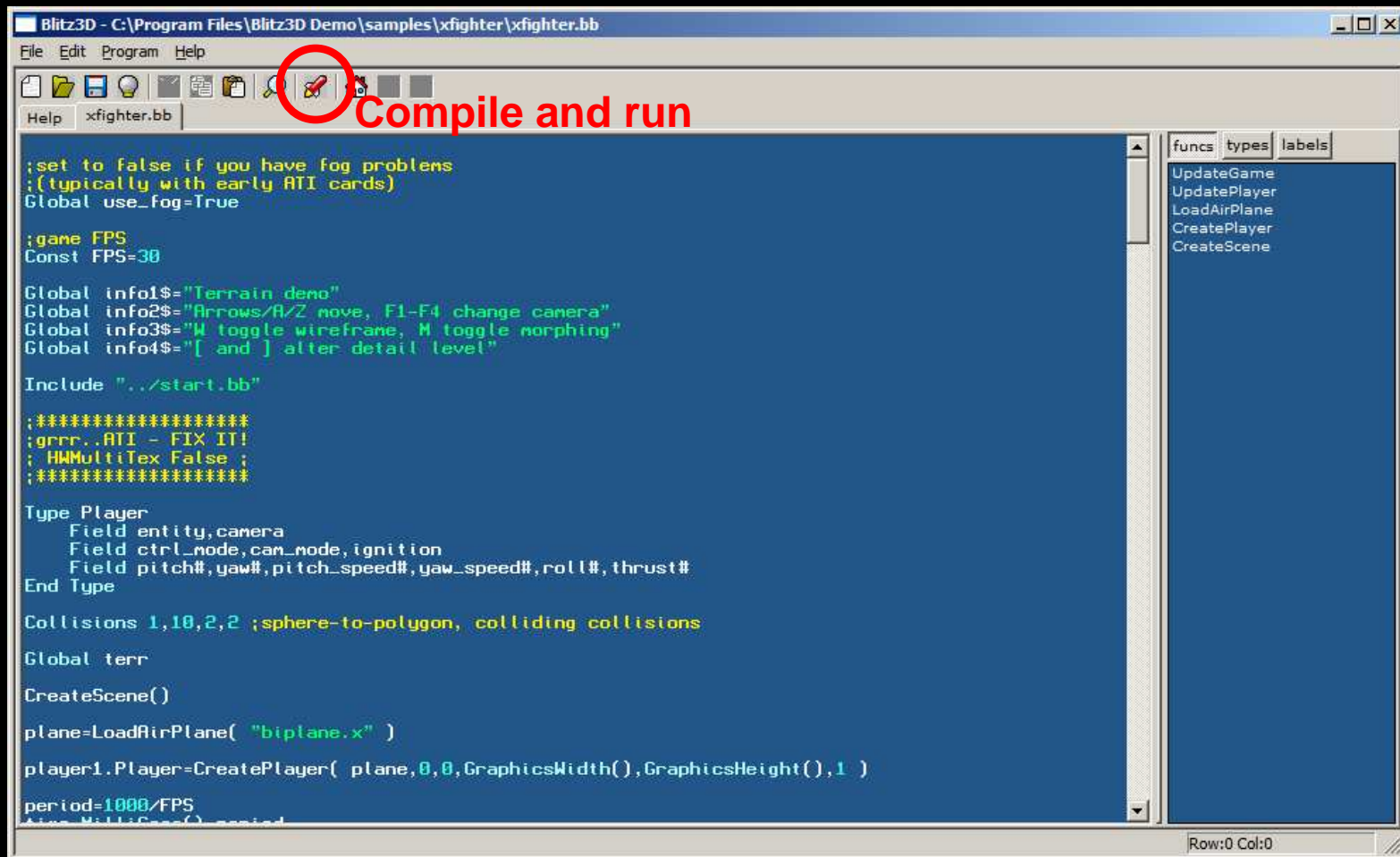
period=1000/FPS
type MultiScene()
;*****
```

funcs	types	labels
UpdateGame		
UpdatePlayer		
LoadAirPlane		
CreatePlayer		
CreateScene		

Row:0 Col:0

Blitz3D

- Another simple IDE



The screenshot shows the Blitz3D IDE interface. The title bar reads "Blitz3D - C:\Program Files\Blitz3D Demo\samples\xfighter\xfighter.bb". The menu bar includes "File", "Edit", "Program", and "Help". The toolbar contains several icons, with a red circle highlighting the "Compile and run" icon (a red lightning bolt). The main text area contains the following code:

```
;set to false if you have fog problems
;(typically with early ATI cards)
Global use_fog=True

;game FPS
Const FPS=30

Global info1$="Terrain demo"
Global info2$="Arrows/A/Z move, F1-F4 change camera"
Global info3$="W toggle wireframe, M toggle morphing"
Global info4$="[ and ] alter detail level"

Include "../start.bb"

;*****
;grrr..ATI - FIX IT!
;HMMultiTex False;
;*****

Type Player
  Field entity,camera
  Field ctrl_node,can_node,ignition
  Field pitch#,yaw#,pitch_speed#,yaw_speed#,roll#,thrust#
End Type

Collisions 1,10,2,2 ;sphere-to-polygon, colliding collisions

Global terr

CreateScene()

plane=LoadAirPlane( "biplane.x" )

player1.Player=CreatePlayer( plane,0,0,GraphicsWidth(),GraphicsHeight(),1 )

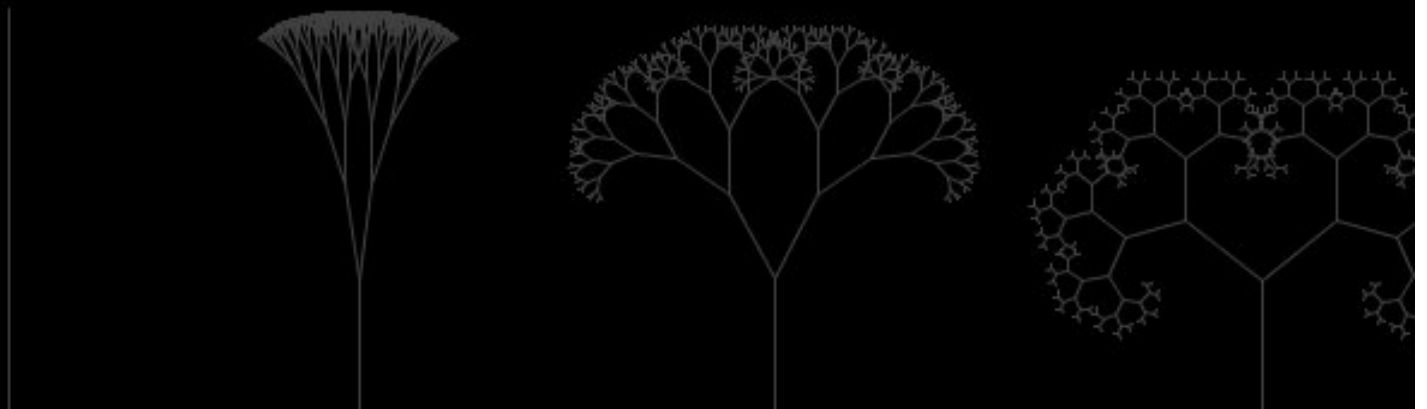
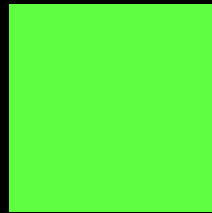
period=1000/FPS
Atx=MultiTex( ) ;noted
```

On the right side, there is a panel with tabs for "funcs", "types", and "labels". The "funcs" tab is active, showing a list of functions: UpdateGame, UpdatePlayer, LoadAirPlane, CreatePlayer, and CreateScene. The status bar at the bottom right indicates "Row:0 Col:0".

Blitz3D

- Creating color
 - RGB only

`Color 95,255,64` =



Blitz3D

- Creating forms

```
; 2d rectangle  
Rect x1, y1, x2, y2
```

```
; 3D sphere  
CreateSphere()
```

```
; Create 3D mesh on the fly  
mesh = CreateMesh()  
surf = CreateSurface(mesh)  
v0 = AddVertex (surf, -5,-5,0, 0,0)  
v1 = AddVertex (surf, 5,-5,0, 1,0)  
v2 = AddVertex (surf, 0, 5,0, 0.5,1)  
tri = AddTriangle (surf,v0,v2,v1)
```

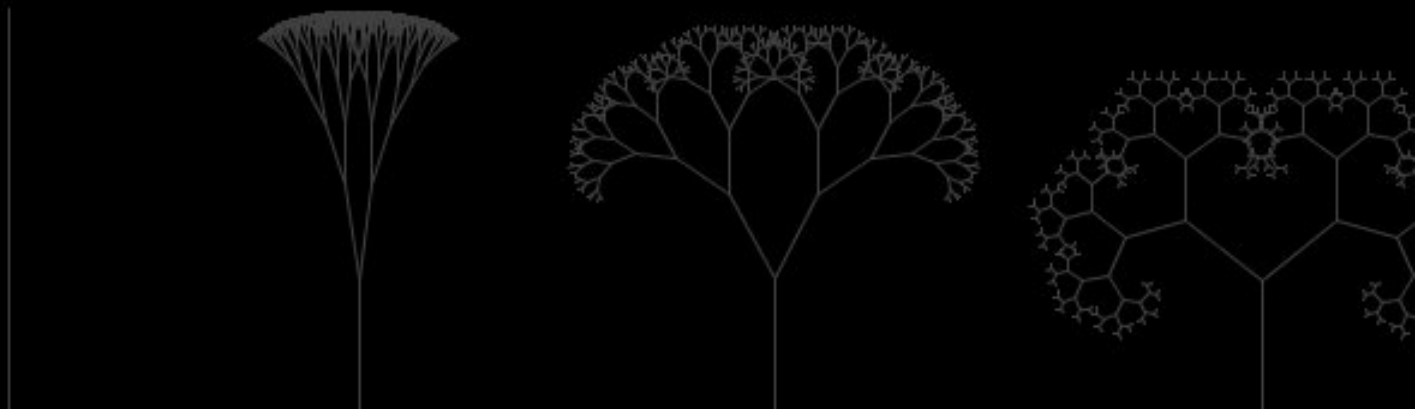


Blitz3D

- Capturing input

```
If KeyDown(203) turn=5  
If KeyDown(205) turn=-5
```

```
If KeyHit(17)  
    wire = Not wire  
    WireFrame wire  
EndIf
```



Blitz3D

- And again, scene setup and a rendering loop

```
;setup lighting
l=CreateLight()

;create terrain
terr=LoadTerrain( "hmap_1024.bmp" )
EntityType terr,10

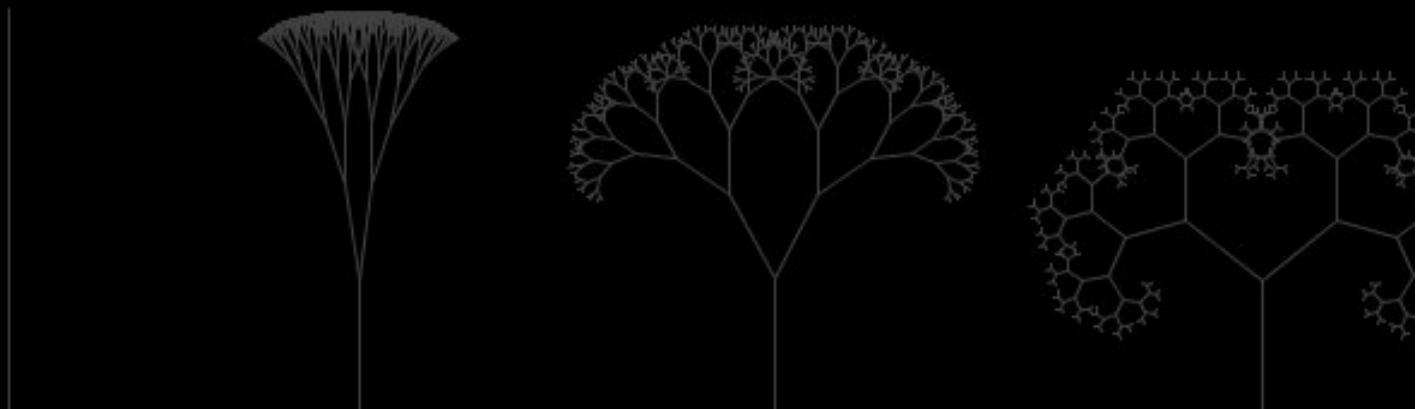
; rendering loop
While Not KeyHit(1)      ; while [esc] is not hit
    ; do stuff here
    UpdateWorld          ; update entity position/collision check
    RenderWorld          ; render to back buffer
    Flip                 ; swap front and back buffer
Wend
```



Blitz3D

- Help is always available!

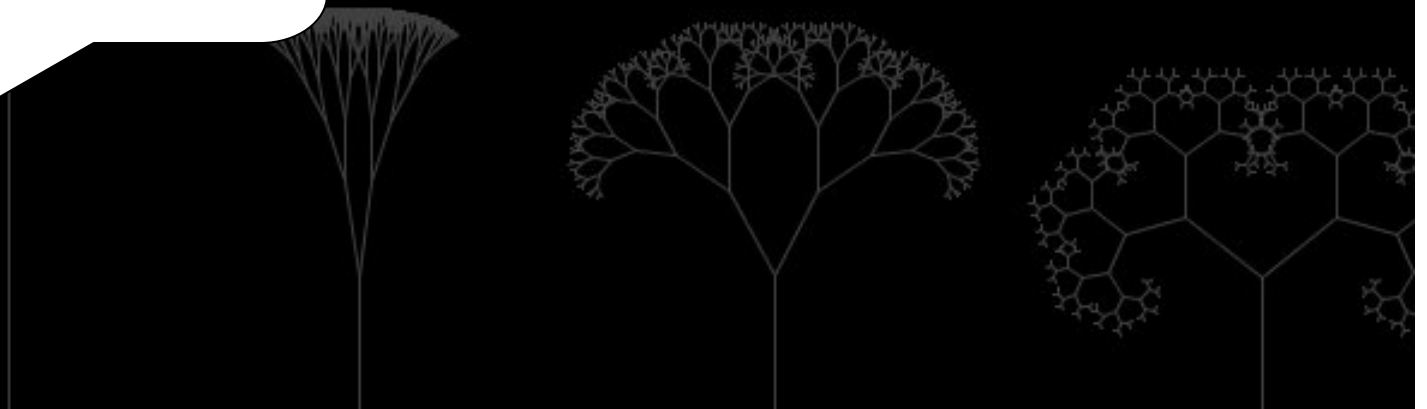
<http://blitzbasic.com/b3ddocs/docs.php>



Grand Finale: Ogre3D



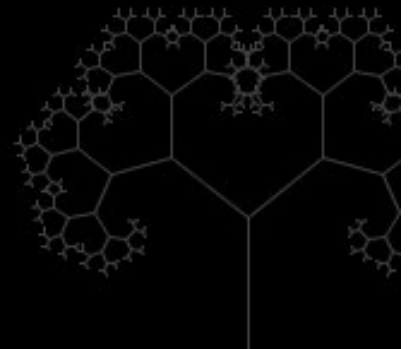
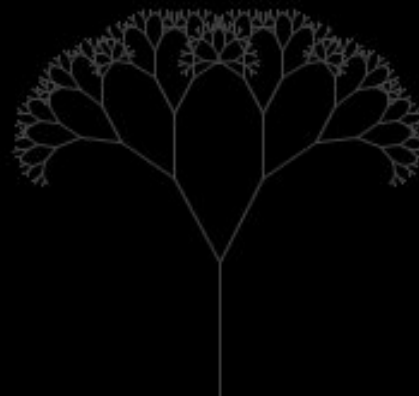
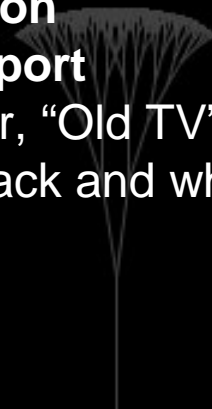
**Hold on to
your butts...**



Ogre3D

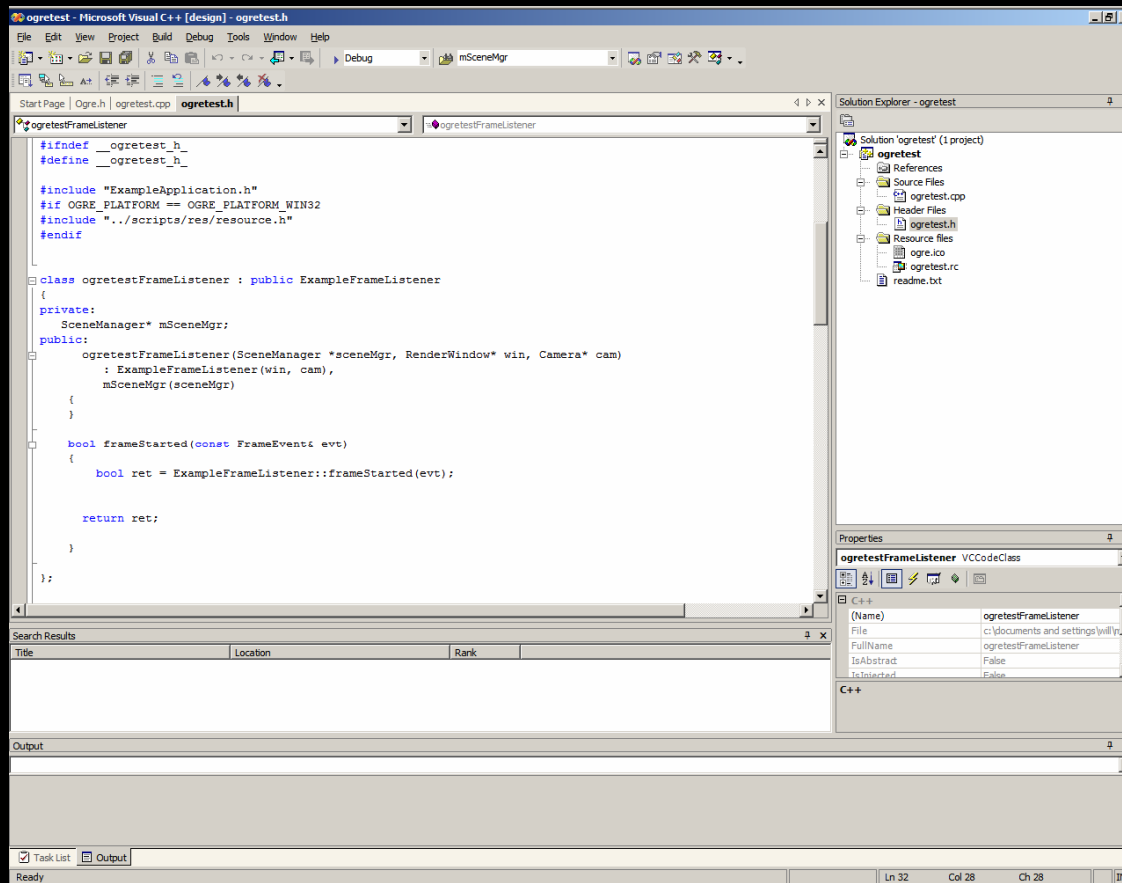
<http://www.ogre3d.org>

- Scripting language: Python
- Full access to source (C++)
- Shader support (cel)
- Animation: bone & vertex
- BSP/LOD
- Reflections/refractions
- Particles
- Maps: normal, bump
- Realtime soft shadows
- Import from: 3DS Max, Maya, Blender, Cinema 4D, XSI, Sketchup, Truespace
- **Realtime texture projection**
- **Multipass rendering support**
 - heat vision motion blur, “Old TV”, HDRI, bloom, invert, black and white



Ogre3D

- IDE: Your choice



- Microsoft Visual Studio
- Eclipse
- GCC
- etc.

Ogre3D

- Once again, scene setup and a rendering

```
// preliminary setup
mSceneMgr = mRoot->createSceneManager(ST_GENERIC,
    "ExampleSMInstance");
mCamera = mSceneMgr->createCamera("PlayerCam");
mCamera->setPosition(Vector3(0,0,500));
```

```
virtual void createScene(void)
{
    // add your geometry here
}
```

```
int main(int argc, char *argv[])
{
    mRoot->startRendering(); // rendering loop
}
```



Ogre3D

- The results can be beautiful



The Magic of Stonehenge
Arsen Gnatkivsky
382 Lines

