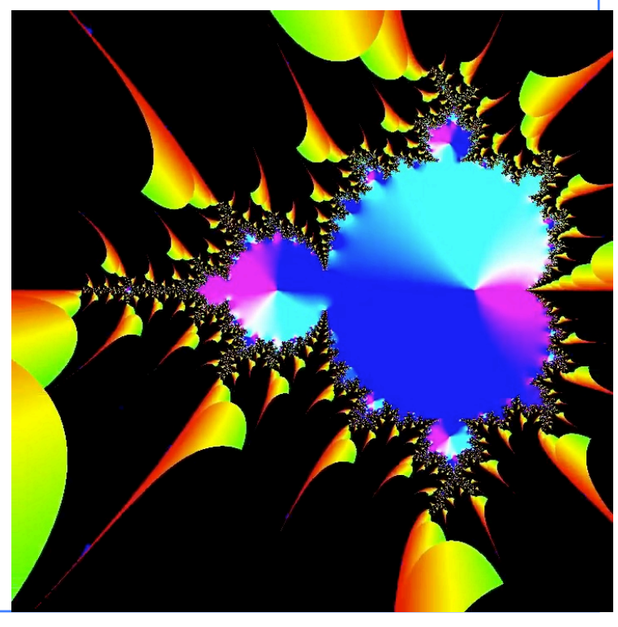
TSBK07/TSBK11

Computer Graphics

Ingemar Ragnemalm, ISY



Lecture 13

Fractals

Low-level algorithms

Other platforms

Monday: Dugga retake

Any and all you want!

You can not lower your score!

Lecture questions

1. What defines a fractal?

2. What does 2.5 dimensions mean?

3. How does L-systems work?

4. How can L-systems be applied to fractals?

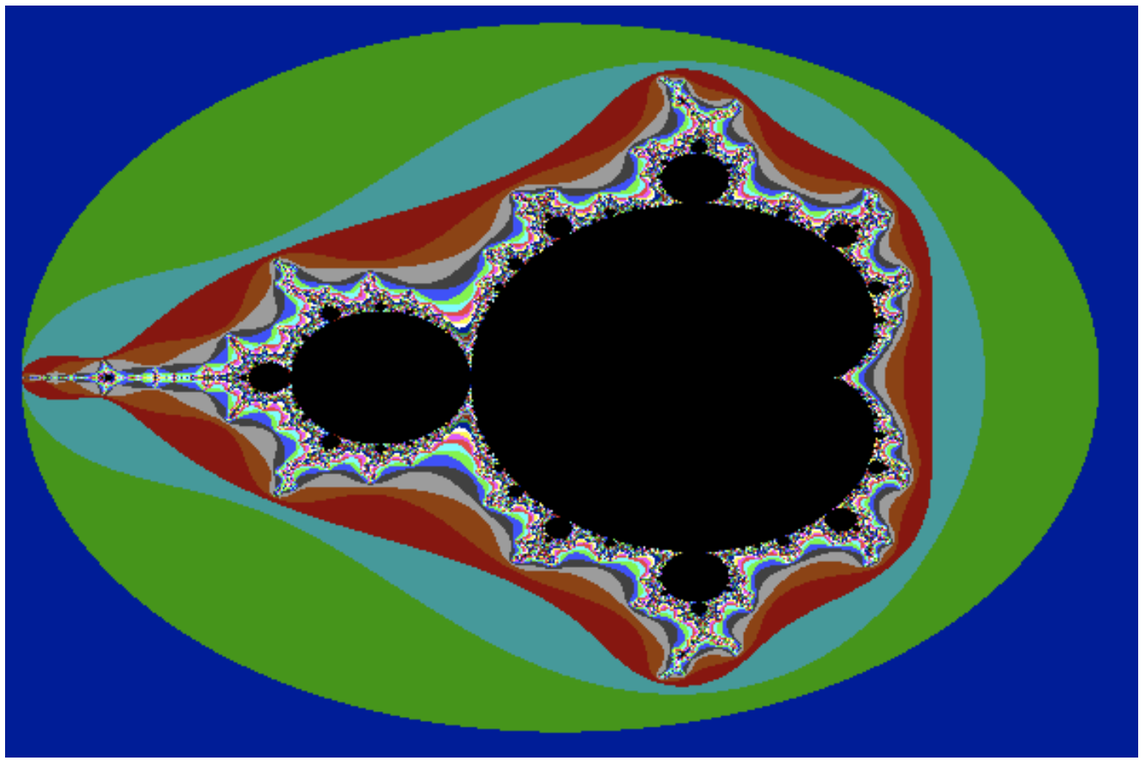
5. How can you draw a 2D geometric fractal?

6. When running a self-squaring fractal, what is the output?

Fractals

Creating complex and interesting shapes from code

Most famous fractal: Mandelbrot set

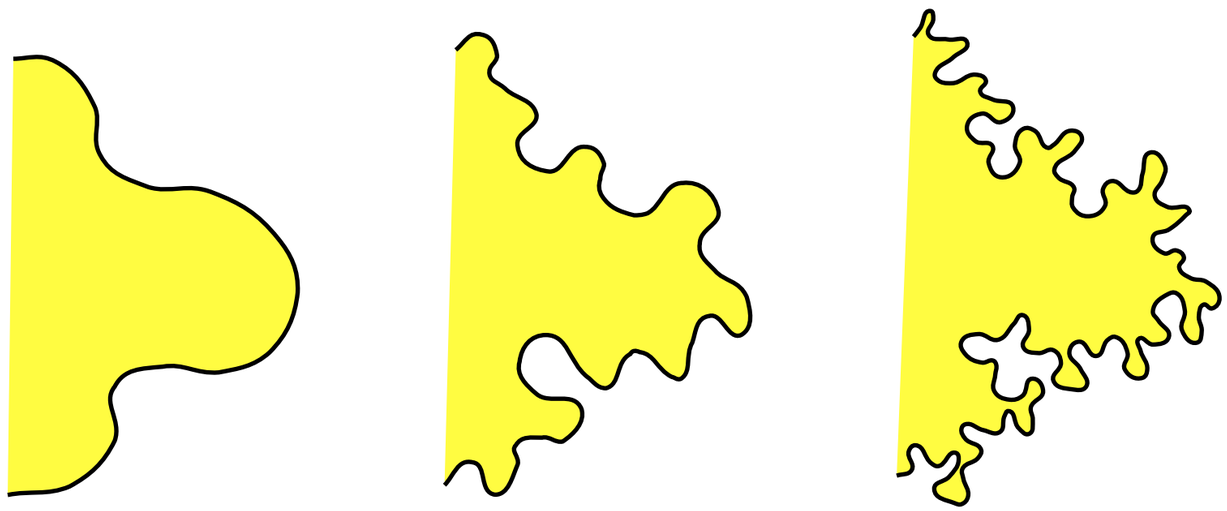


What is it, more than a pretty image?

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

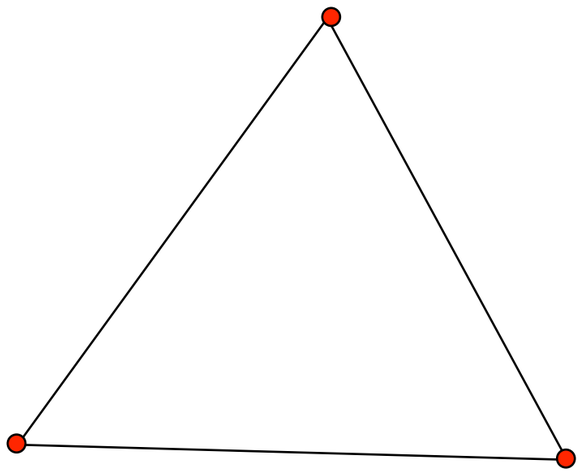
Used for modelling such shapes

Classification of fractals

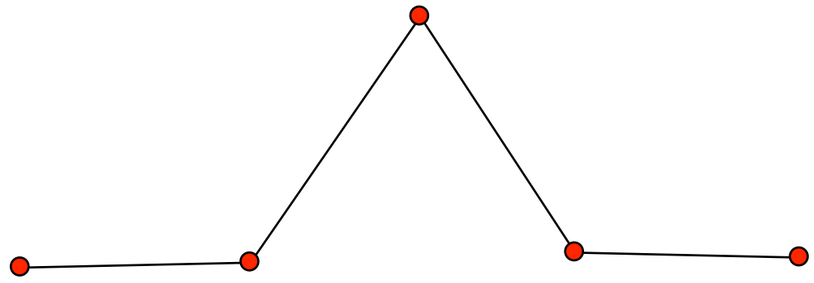
- geometrical recursive construction
- stochastic fractals
- mathematical formulas (in the complex plane)

Geometric construction of self-similar fractals

Example: Koch curve

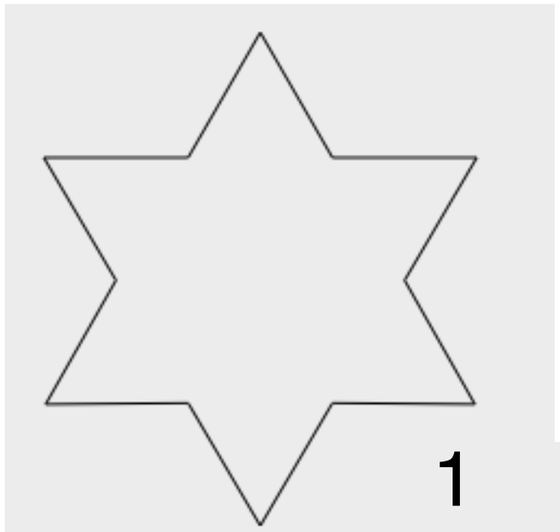


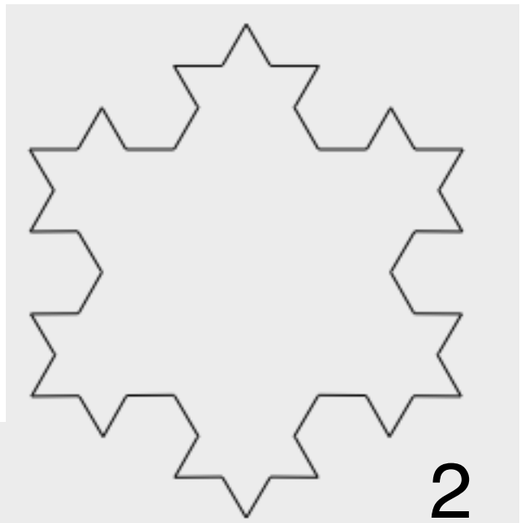
Initiator

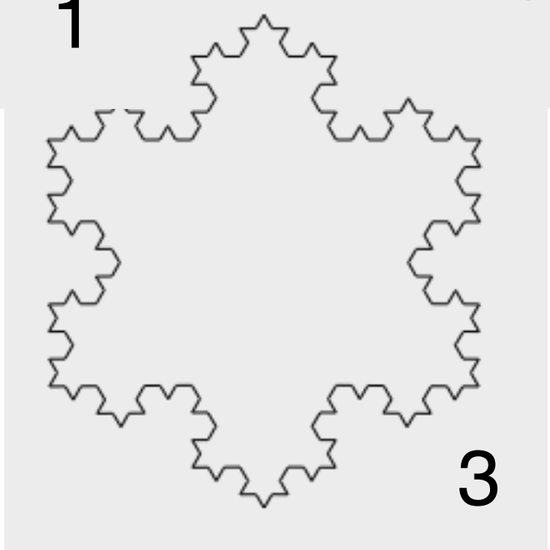


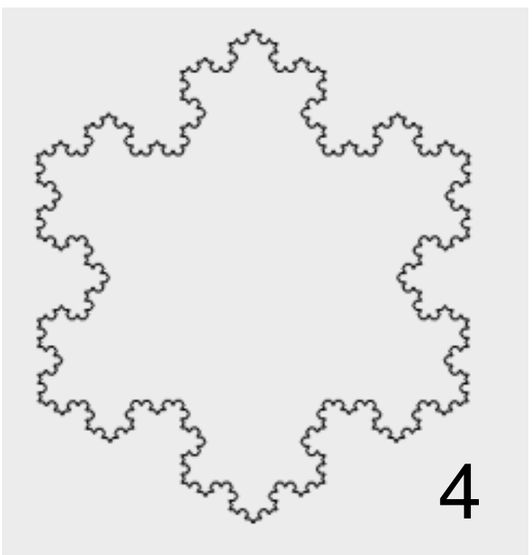
Generator

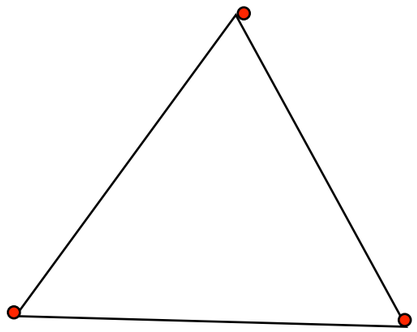
Resulting Koch curves



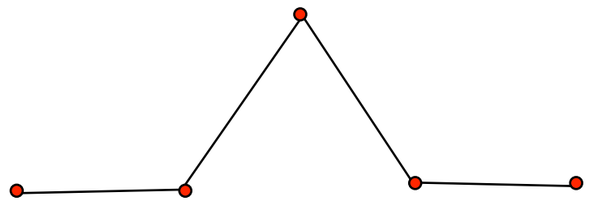








Initiator



Generator

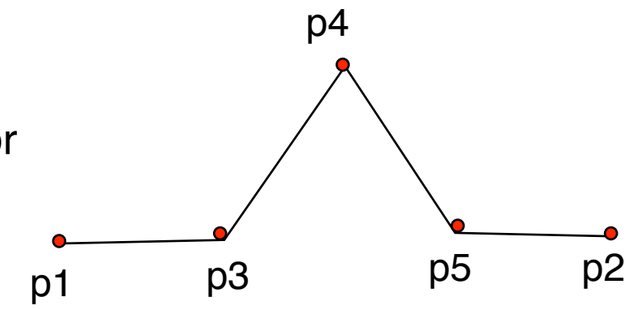
Recursive function

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)

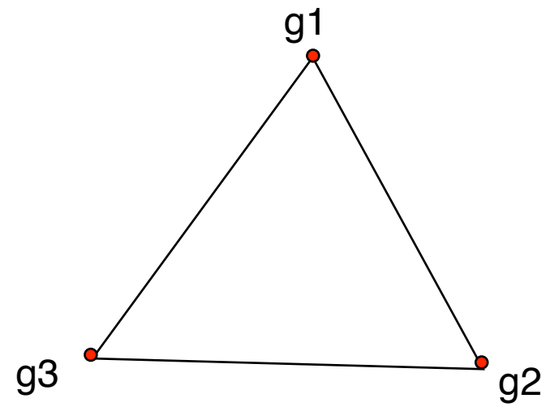
```
procedure DrawKoch(p1, p2, depth)

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:

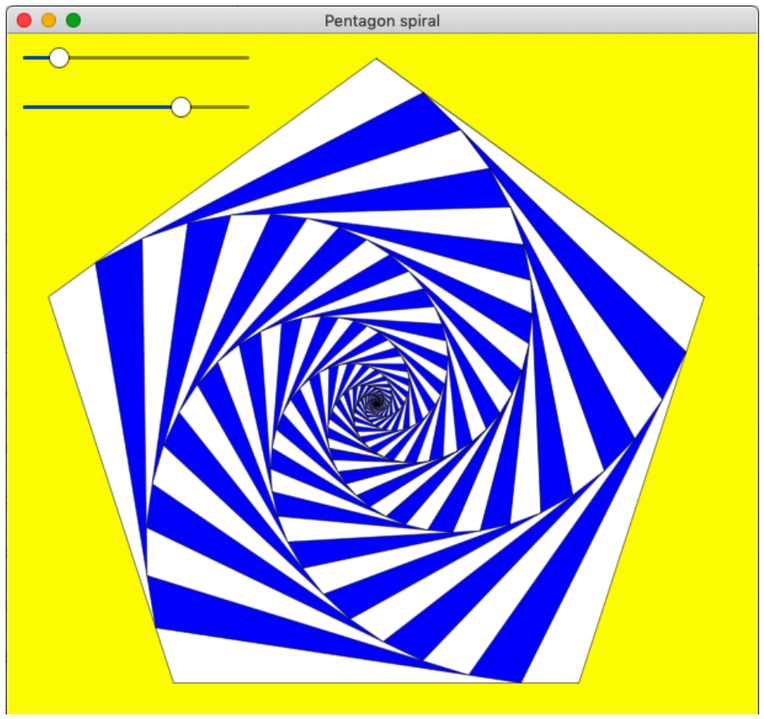
Choose three generator points, g1, g2, g3

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



Embarassingly simple: Just repeat a shape and scale

This just draws polygons on top of each other, rotate and scale.



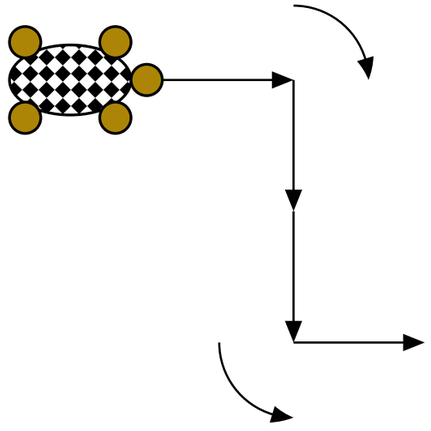
Never underestimate the power of the many!

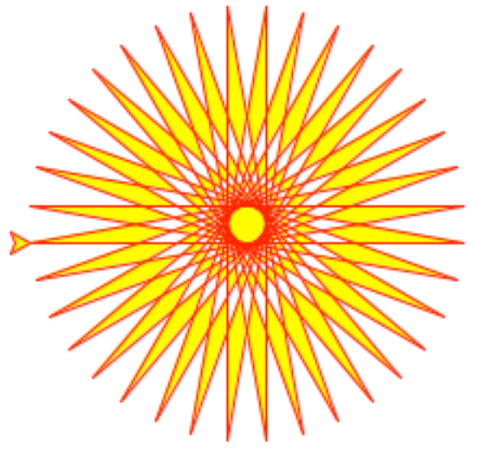
Geometric 2D fractals and Turtle Graphics:

Turtle graphics = local operations

Forward, turn right, turn left, move forward, paint forward

Koch: Forward, turn left, forward, turn right, turn right, forward, turn left, forward

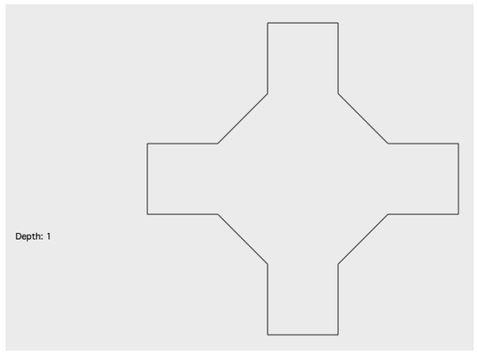


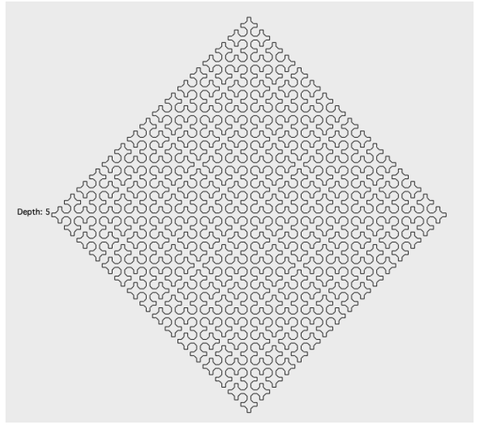


More geometric 2D fractals:

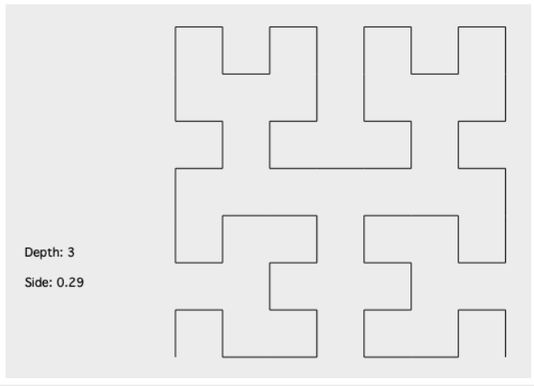
Often defined as turtle graphics

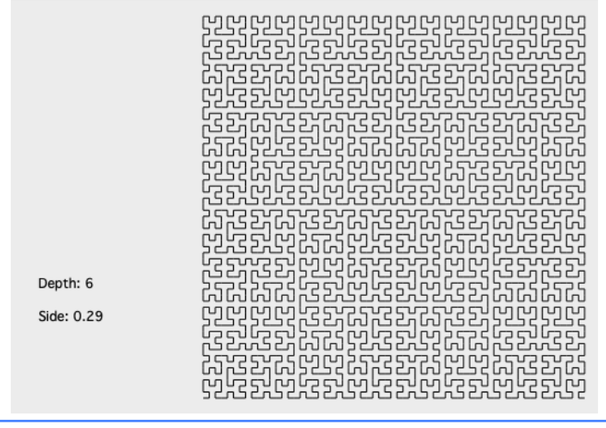
Sierpinski curve



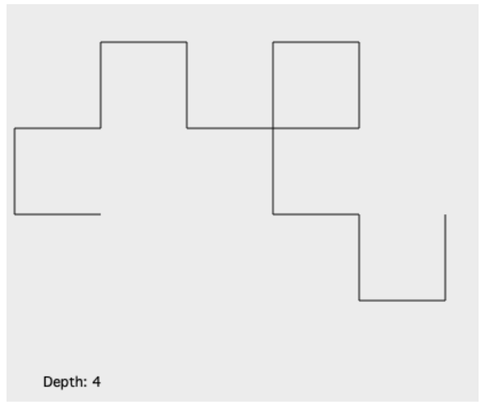


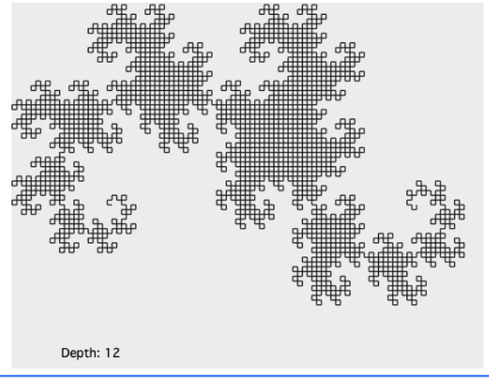
Hilbert curve





Highway dragon





Popular way to describe 2D fractals: L-systems

Based on strings

Definition:

Axiom: The initiator, starting string

Production rules: Replacement of symbols for each step

Example:

Axiom: RALADGDG

Production rules:

A -> AMA
G -> ING

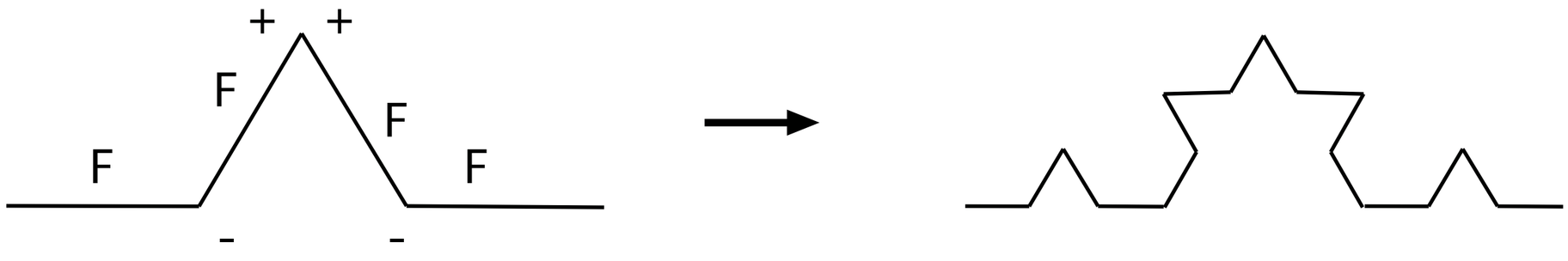
One iteration: RAMALAMADINGDONG
Two iterations: R AMA M AMA L AMA M AMA DIN ING DON ING

Koch curve from L-systems

F → F - F + + F - F

produces a Koch curve if

- = turn left
+ = turn right



Needs a rescaling for each step

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

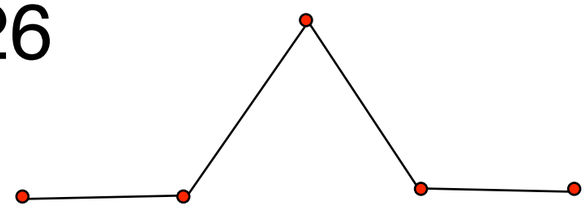
Solves to D = ln(n) / ln (1/s)

Fractal dimension example: Koch curve

n = 4

s = 1/3

D = ln 4 / ln 3 = 1.26



Fractal dimension example: Splitting a line

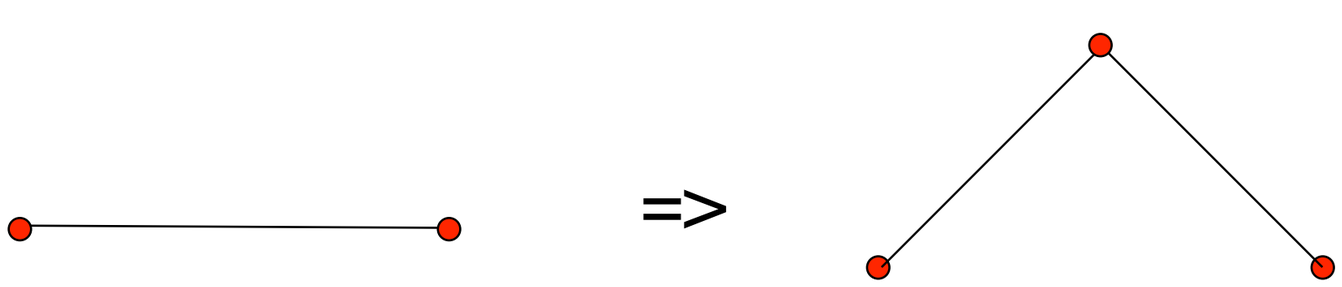
=>

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:

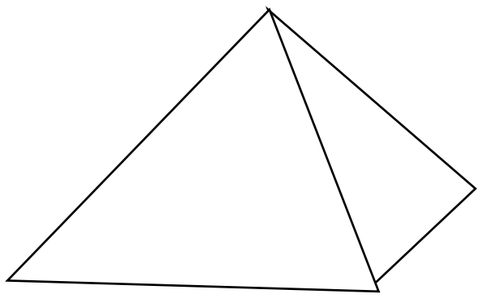
1 to 2: Well-behaved fractal curve, limited length

>2: Self-intersecting, area-covering

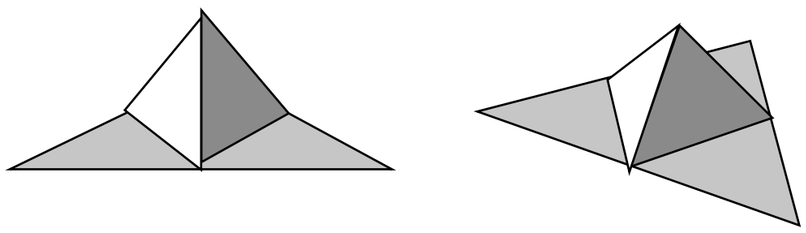
Split line: D = 1 minimum, no fractal
Koch: D = 1.26, moderate fractal
Moved midpoint: D = 2, maximum

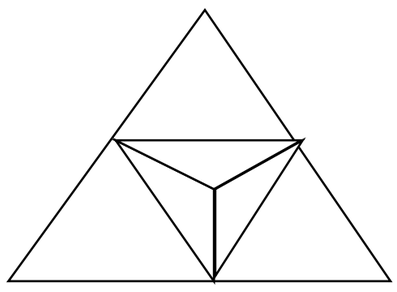
Geometric construction of self-similar fractals in 3D

Example



Initiator





Generator

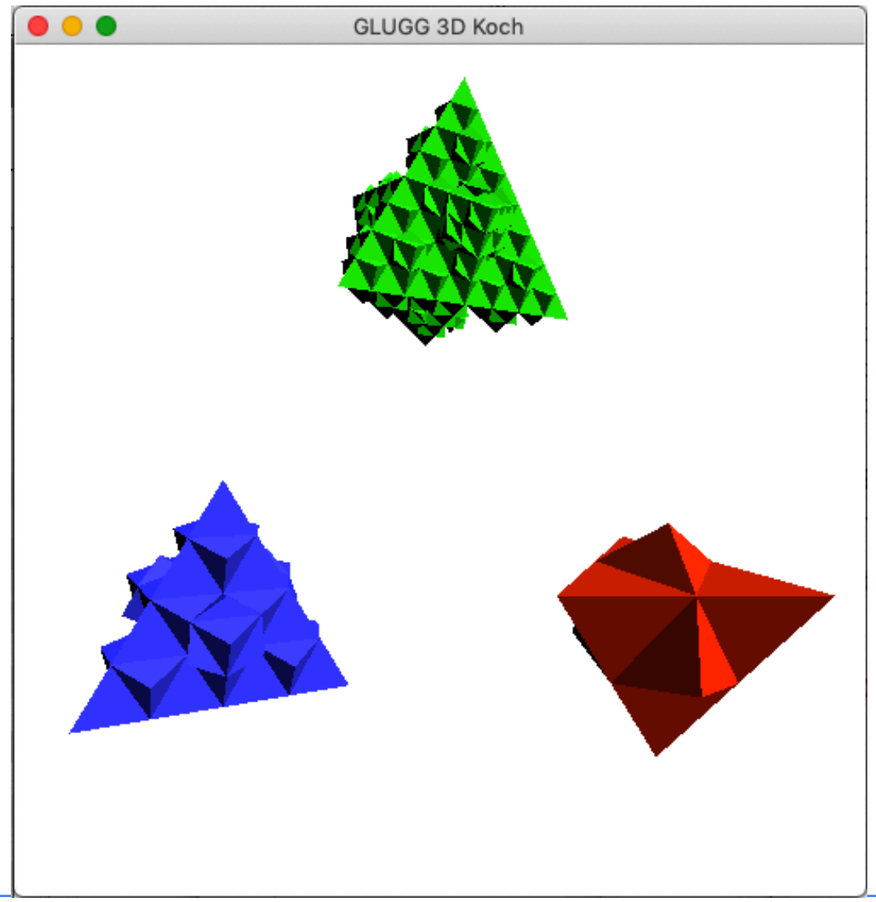
n = 6

s = 1/2

D = ln 6 / ln 2 = 2.58

Koch in 3D

Subdivide triangles, built using GLUGG



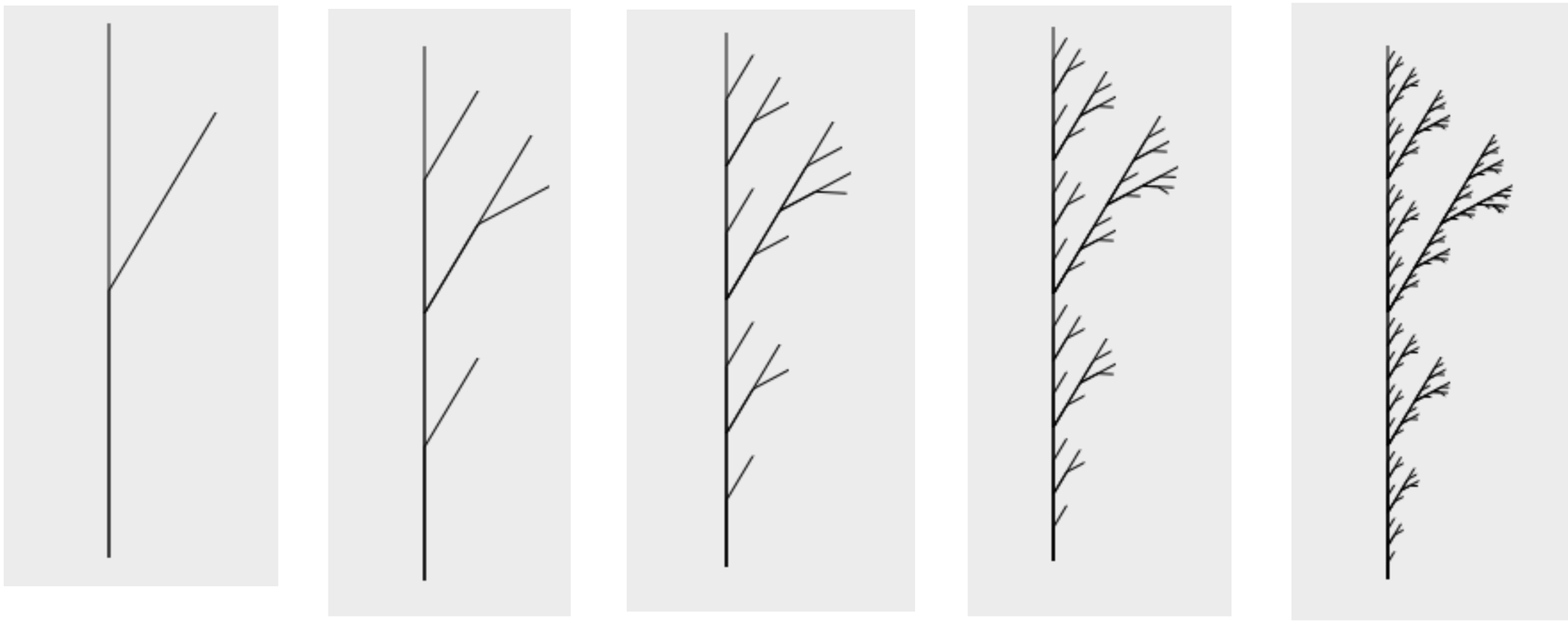
Interpretation of fractal dimension:

In 3D:

2 to 3: Well-behaved fractal surface, limited area

>3: Self-intersecting, volume-covering

Example: Generation of plants



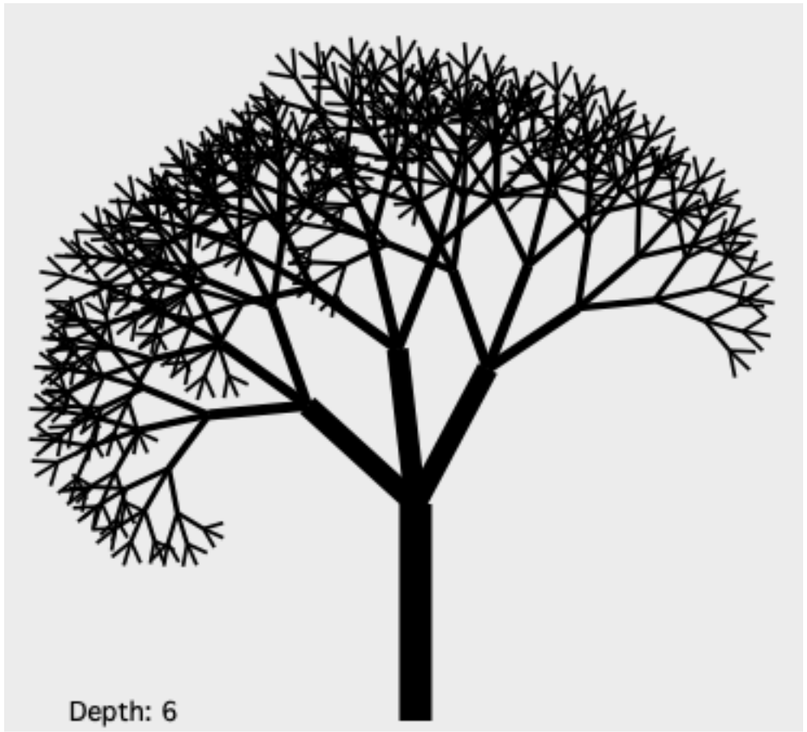
Promising, but too self-similar!

Statistically self-similar fractals

Random variation of generator

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

Example: Generation of plants #2



Related methods:

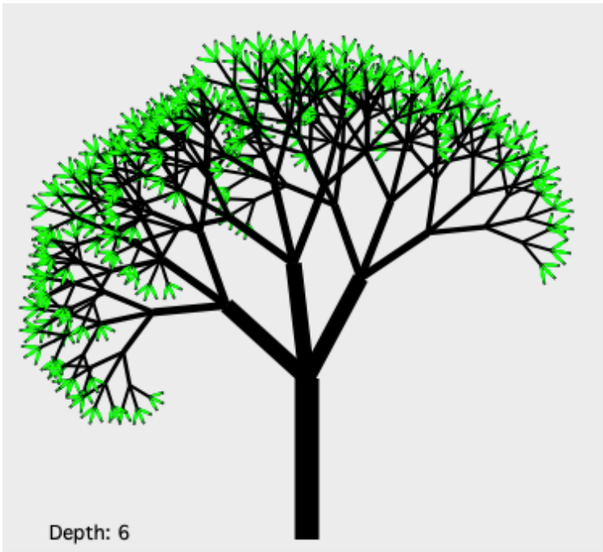
Shape grammars and procedural methods

No unlimited resolution

Different rules at different levels

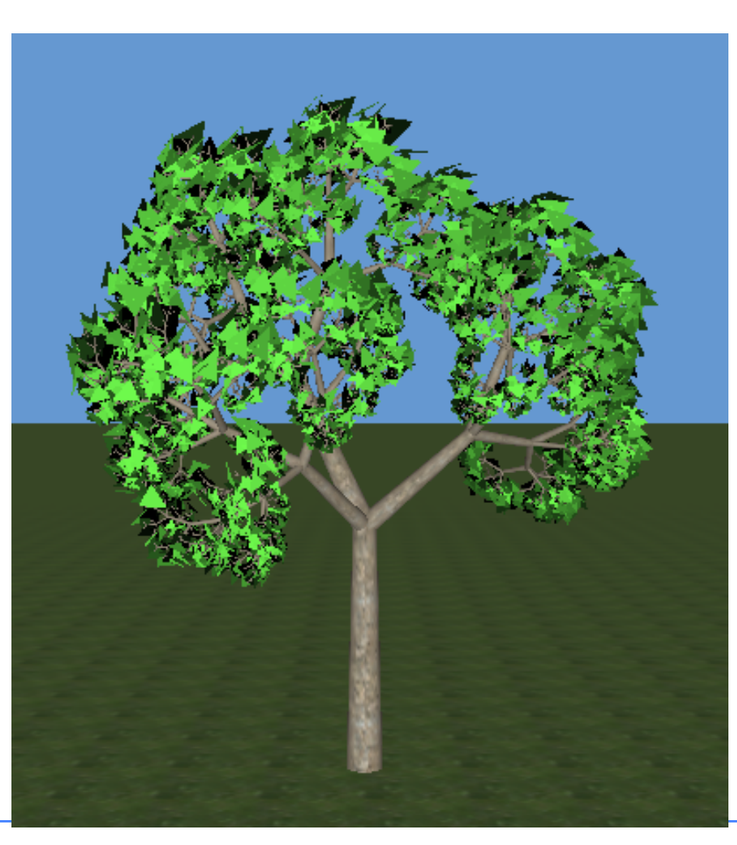
Example: Tree with leaves: replace last iteration with leaf generator

“graftals”



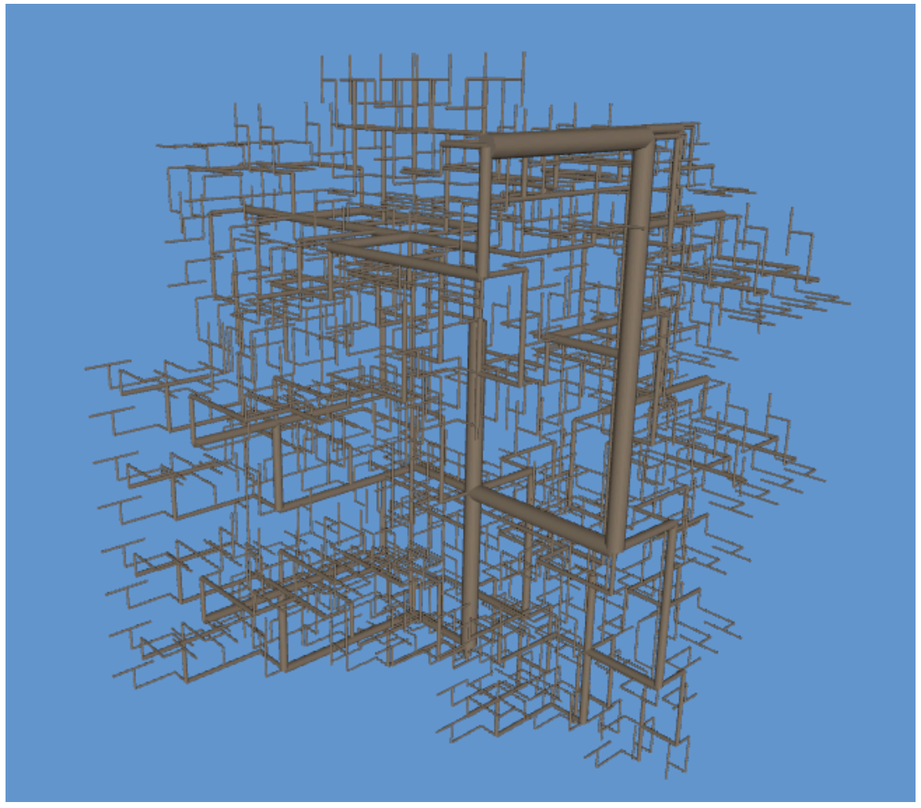
Tree in 3D

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



More geometric 3D fractals:

2D line drawing fractals can be generalized to 3D.



Self-squaring fractals

Based on simple functions in complex space

Insert complex numbers (points) into a function

Apply function recursively, and analyze the behaviour.

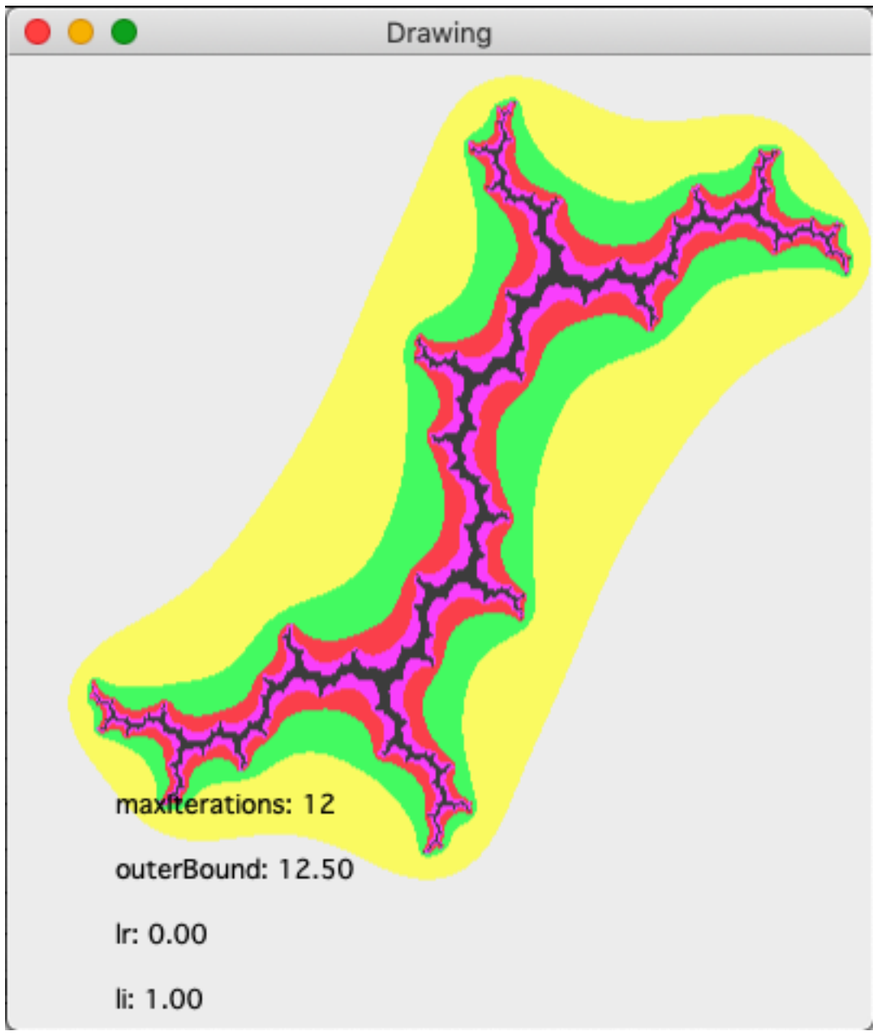
- Diverge?
- Converge?
- Chaotic?

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

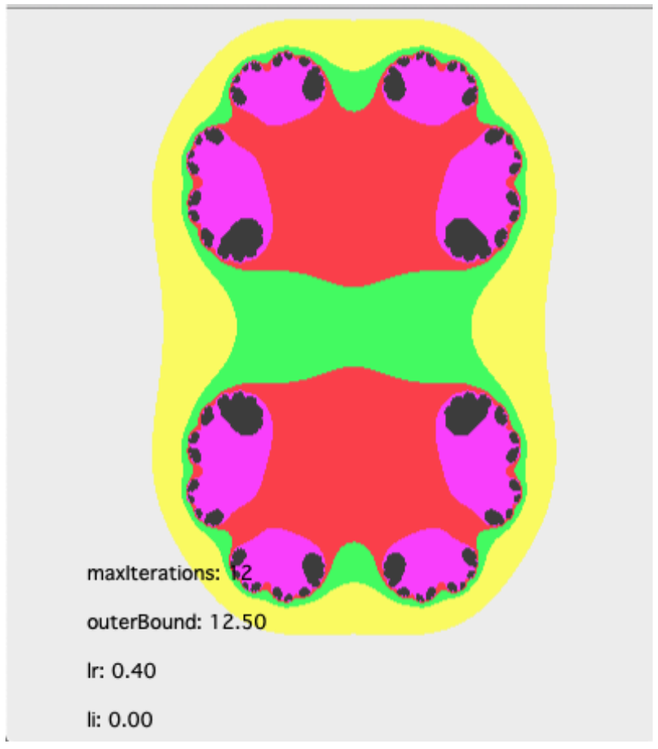
```
for y = miny to maxy
   for x = minx to maxx
      (zr, zi) = scaling of (x,y)

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.

$R^2 \approx 10$

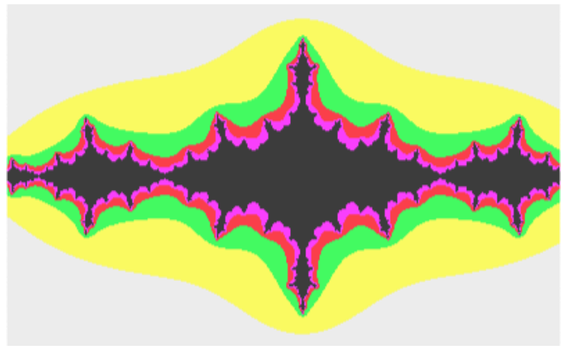
Other Julia sets

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

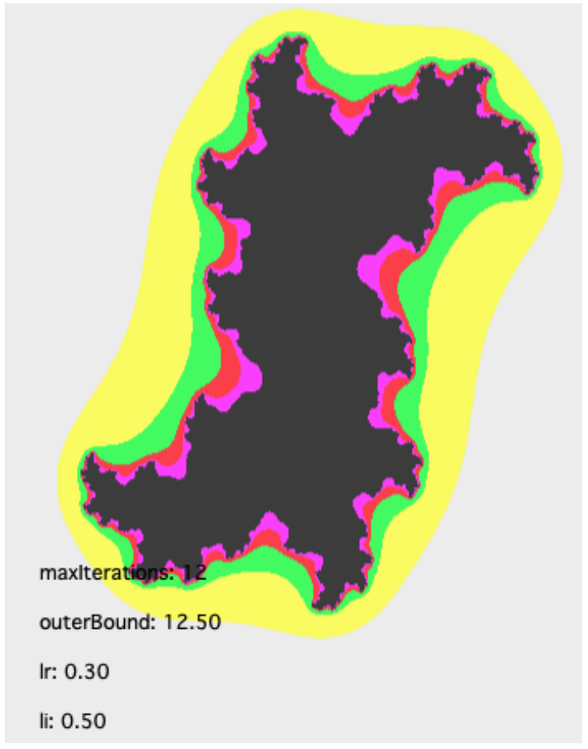


$\lambda = (0.4, 0)$

Other λ values



$\lambda = (-1.3, 0)$

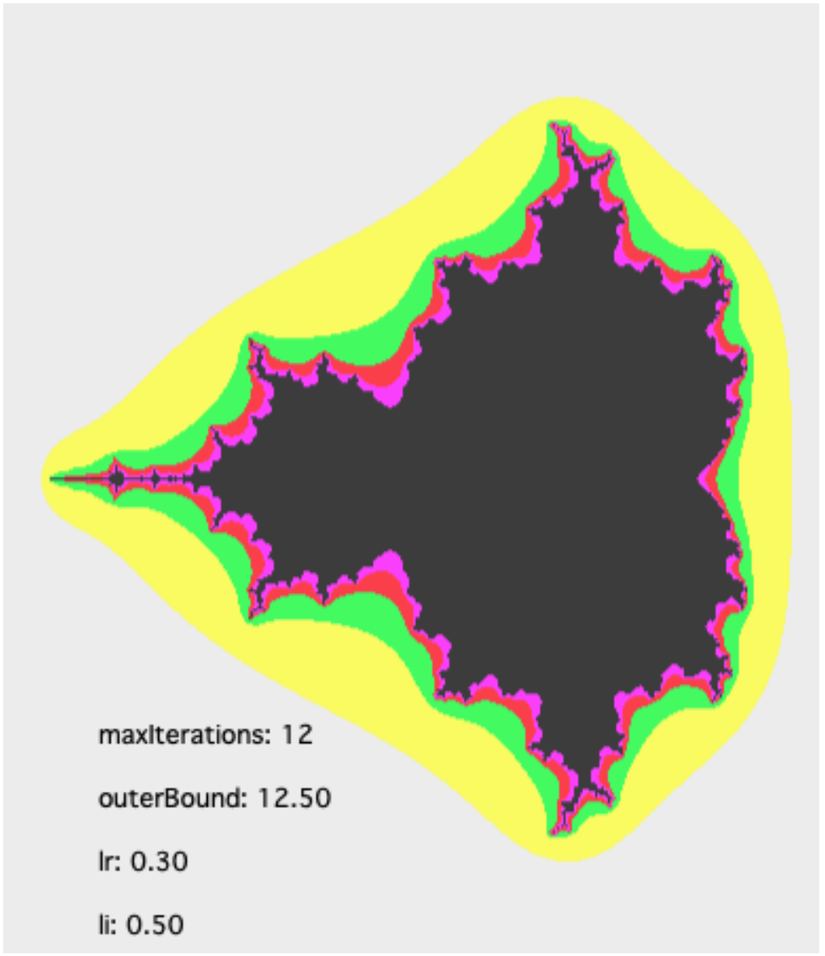


$\lambda = (0.3, 0.5)$

Self-squaring fractals

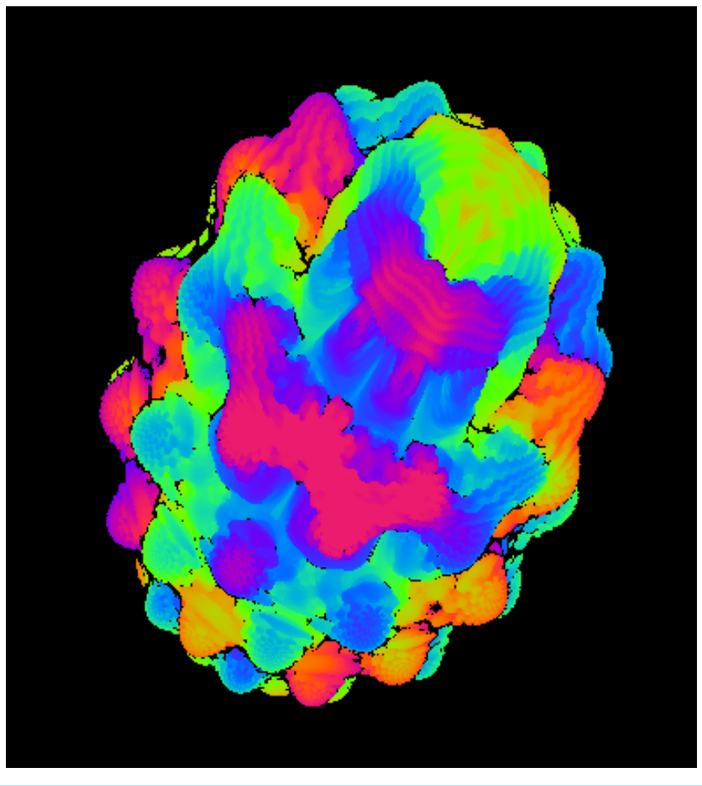
The Mandelbrot

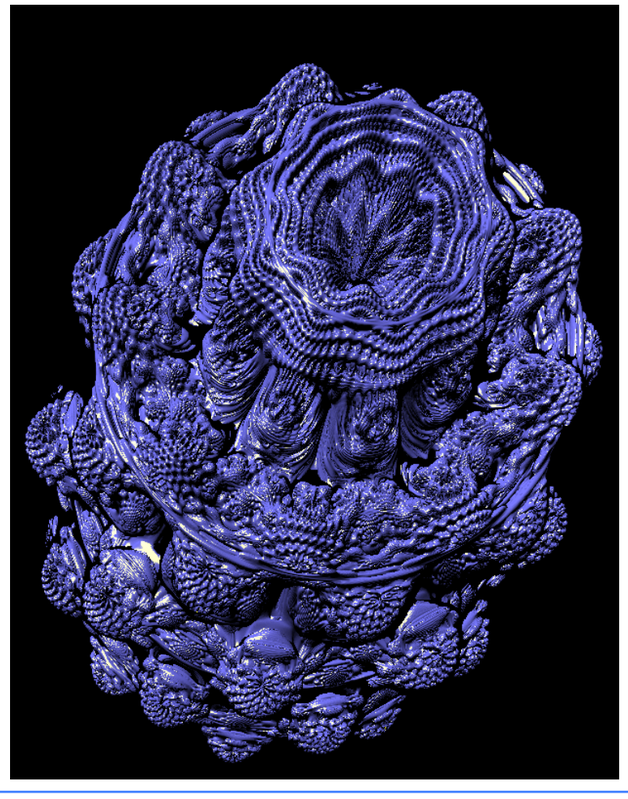
$$z_{k+1} = z_k^2 + z_0$$



More 3D fractals

Mandelbulb. Based on polar coordinates rather than complex numbers.





Mandelbulb

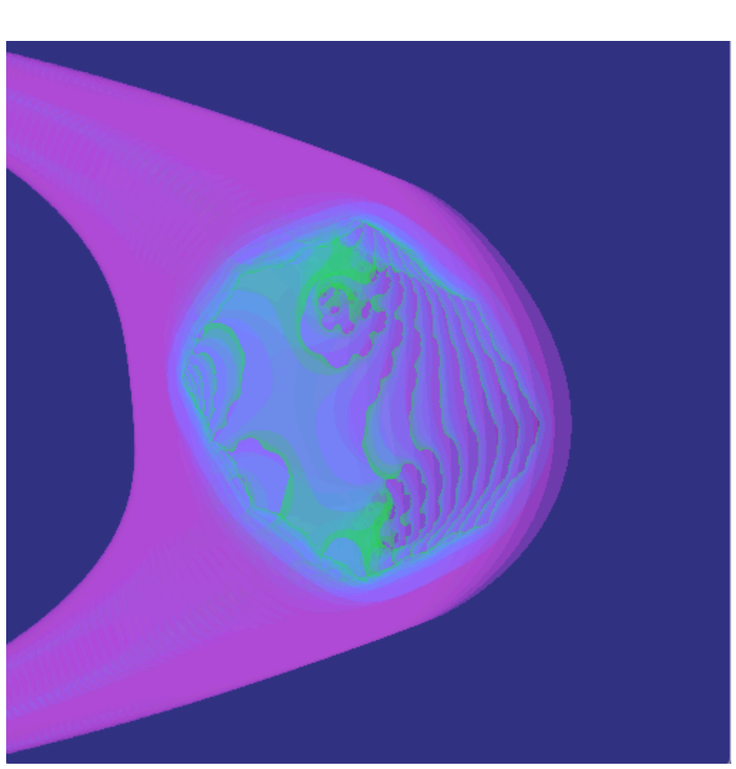
Based on ray-casting.

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

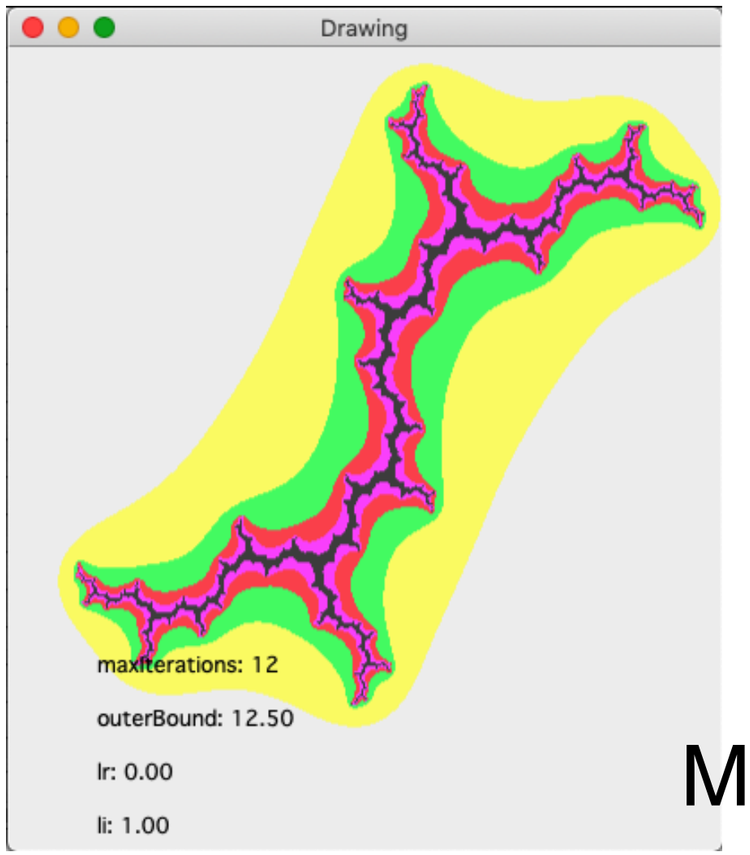
Many other 3D fractals exist.

Multilayer

Take a 2D fractal with some variable parameter. Render with transparency,

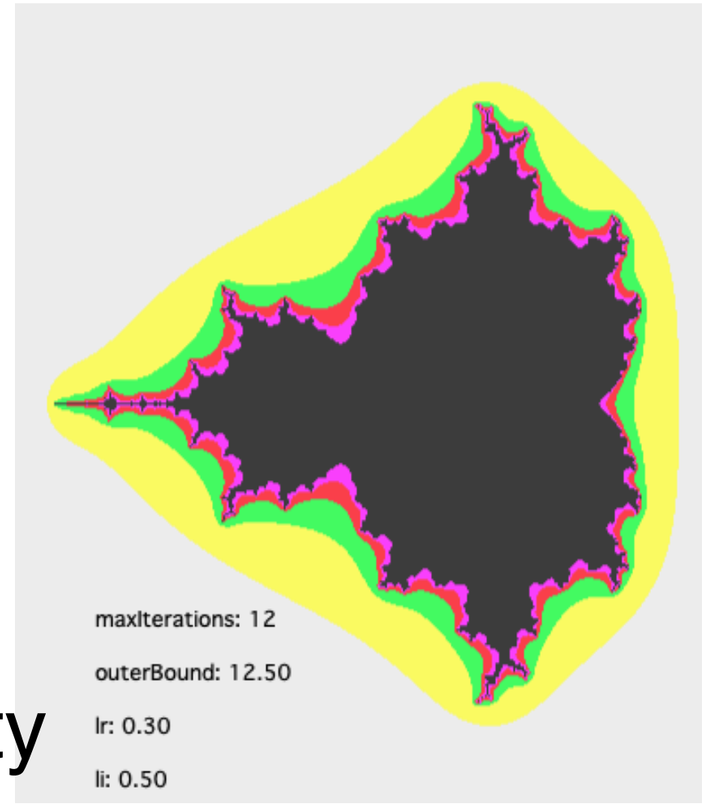


Self-squaring fractals



- 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

More fractals next time

Randomness

Noise

FBM

Fractal terrains

Diffusion

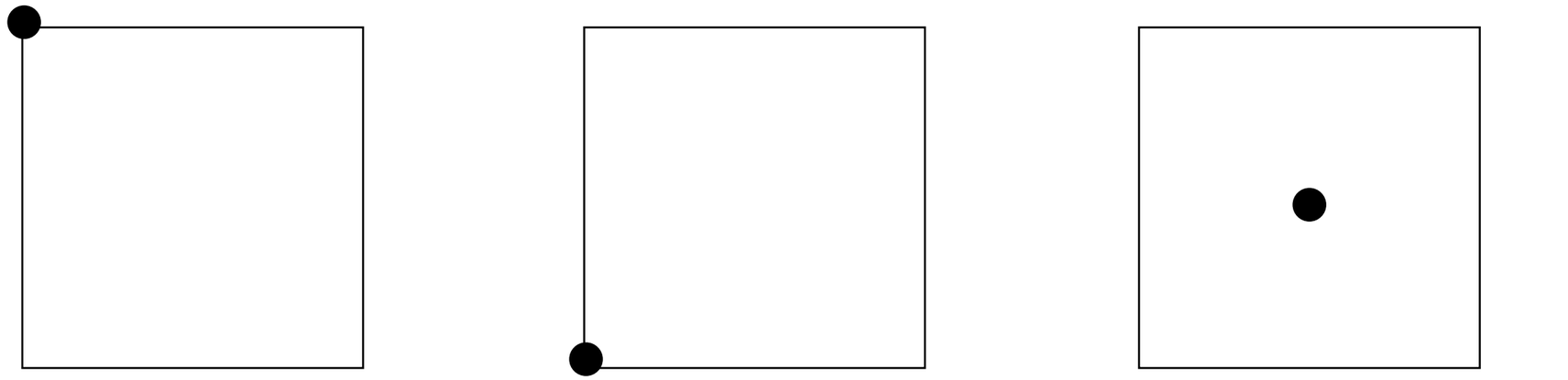
Some low-level issues:

Pixel geometry
Inside-outside tests
Flood fill

Pixel addressing and object geometry

So far, I have been careless with one question:

Where in the pixel (x,y) is the point (x,y)?



Is it here, top-left…?

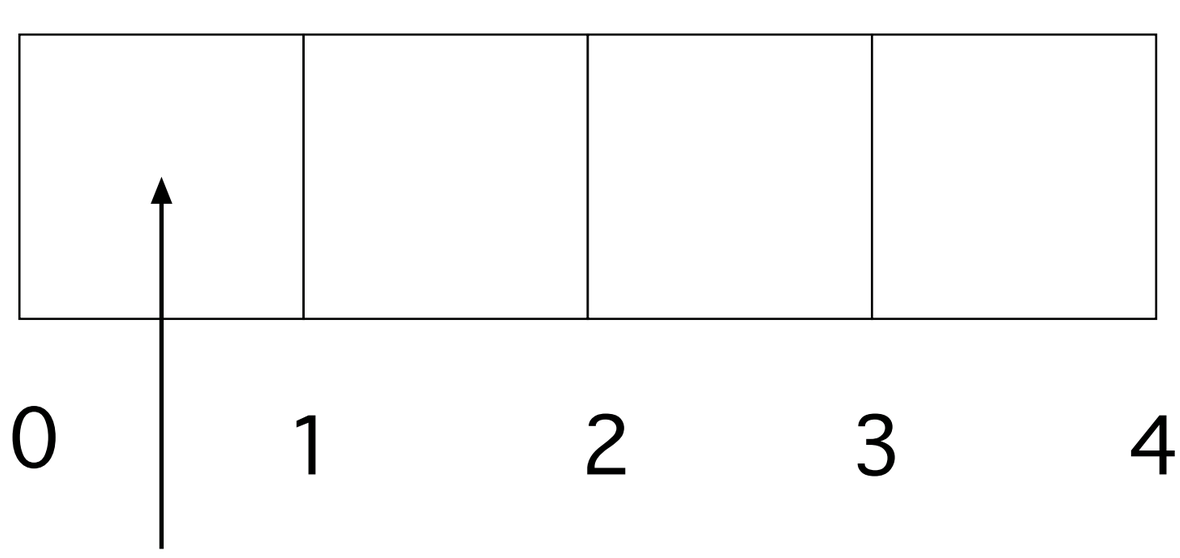
…or here, bottom-left…?

…or perhaps here, centered?

This is the “hotspot” of the pixel

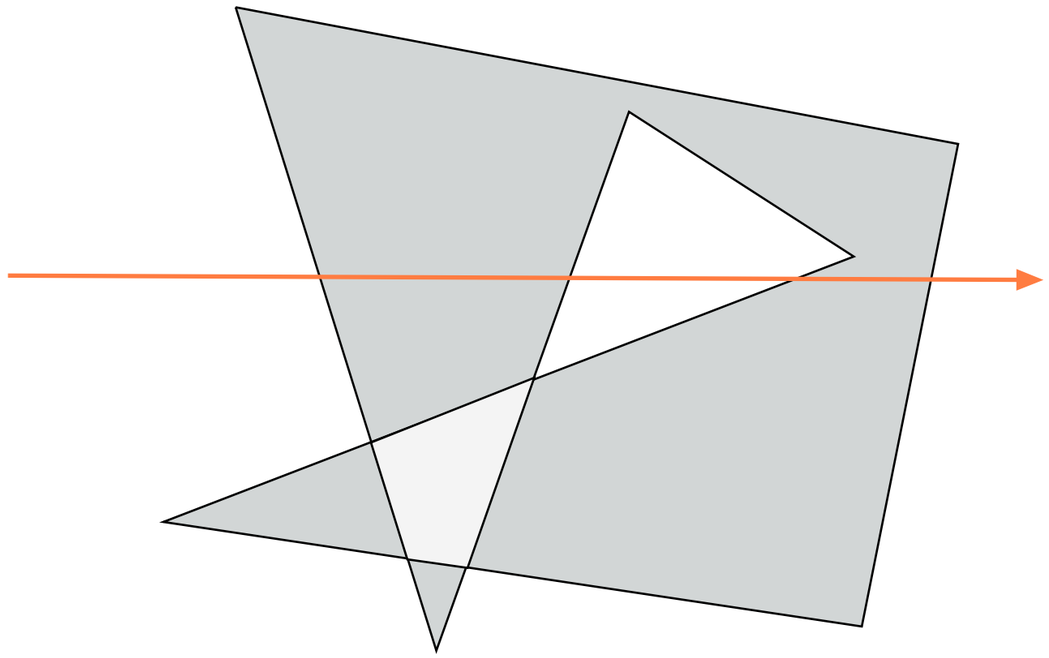
Important case: Texture access

If you use interpolation, which
you usually do.



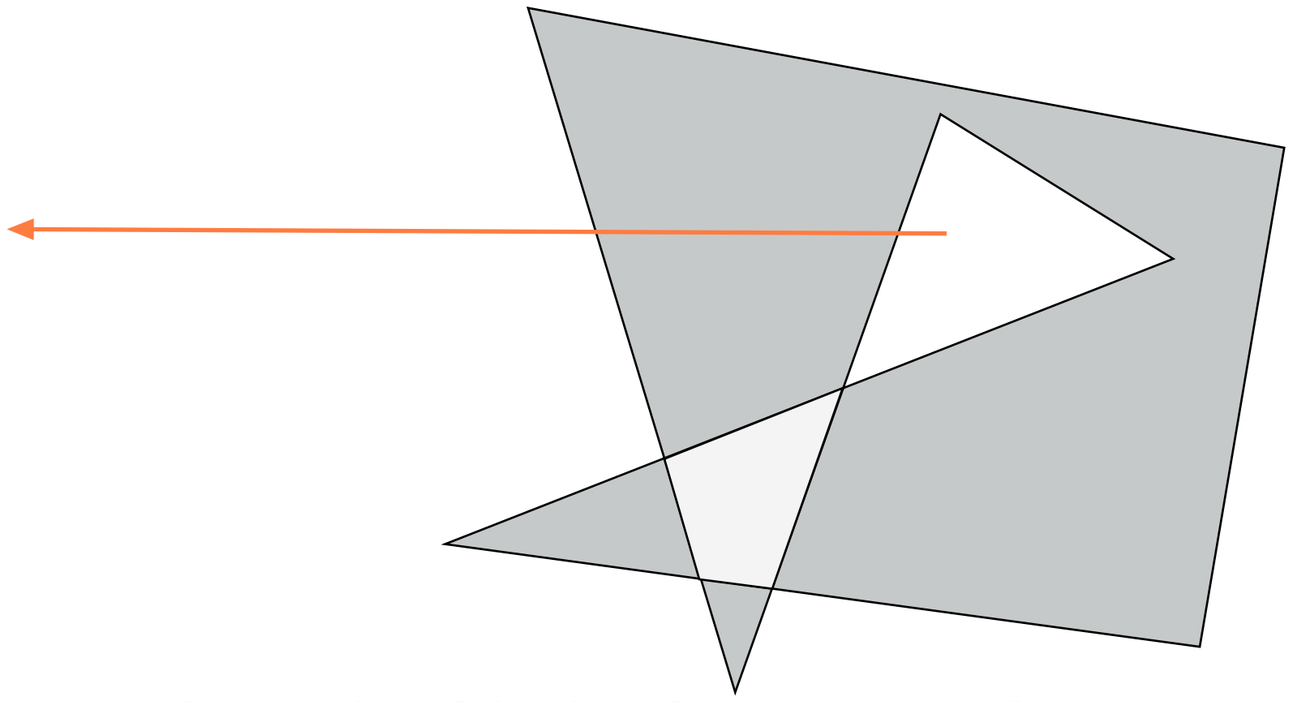
Access at 0.5 to hit the pixel
center = exact pixel value

Scan-line polygon filling



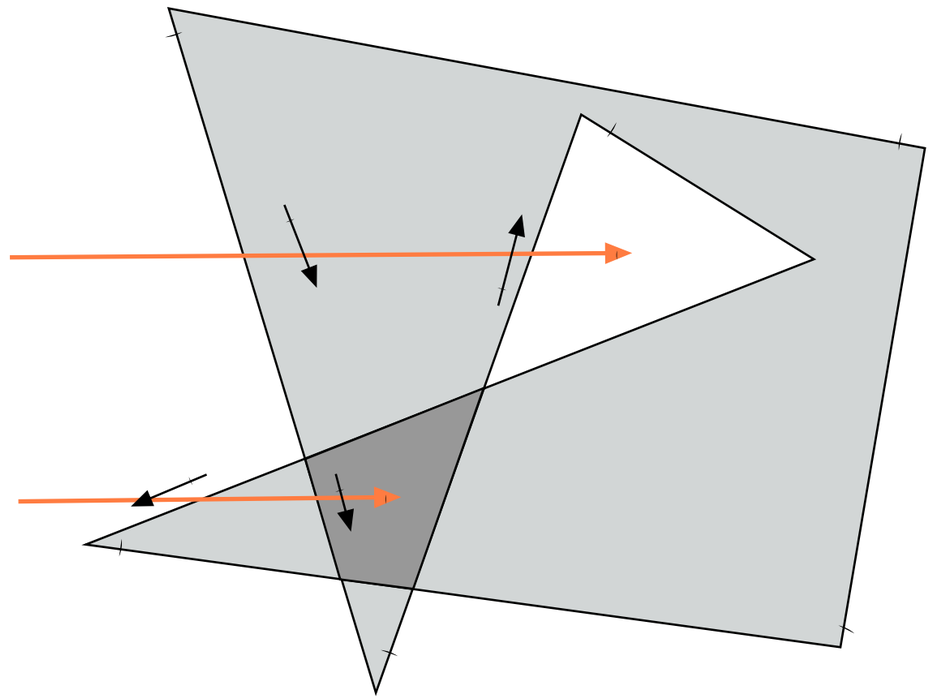
Go from left to right,
fill when there is an odd number
of edges to the left!

Inside-outside tests
Method 1: Odd-even rule



To learn if a pixel is inside the polygon, you can apply the same kind of test!

Inside-outside tests
Method 2: Non-zero winding rule



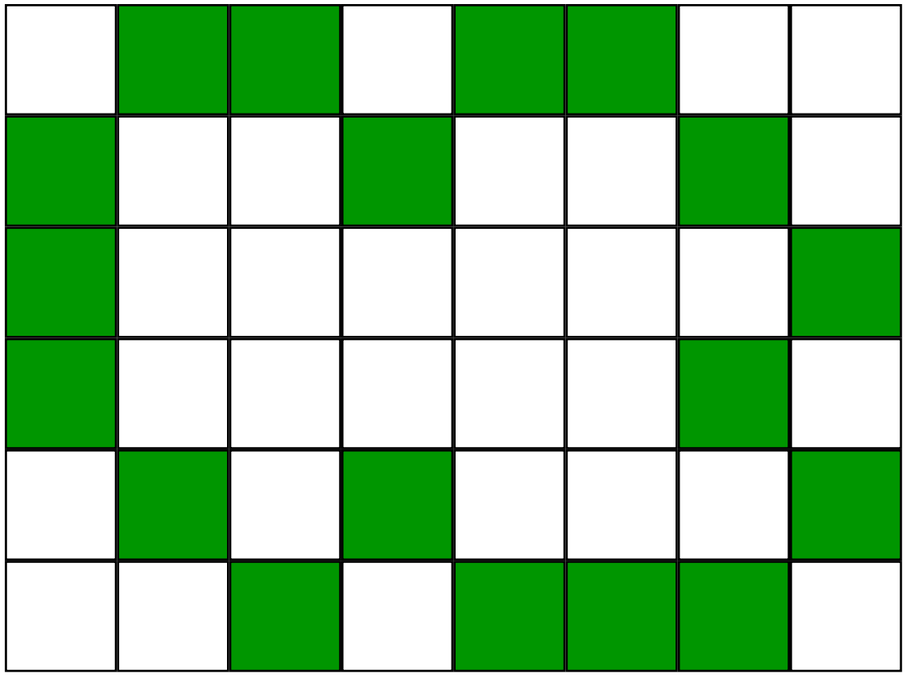
Check the directions of intersections!

Inside-outside tests in 3D

Is a point inside a 3D model?

Same test can be used for closed polyhedra!

Flood fill



A color defines the area to be filled

Simple recursive algorithm:

```
procedure FloodFill(x,y,fill,target);

current := GetPixel(x,y);
if (current = target) then
   SetPixel(x,y,fill);
   FloodFill(x+1, y, fill, target);
   FloodFill(x-1, y, fill, target);
   FloodFill(x, y+1, fill, target);
   FloodFill(x, y-1, fill, target);

procedure StartFloodFill(x, y, fill)
   target := GetPixel(x, y);
   if (fill <> target) then
      FloodFill(x, y, fill, target);
```

Not practical! Too deep recursions!

Flood fill using pixel spans:

```
push starting pixel on stack
while stack not empty
   pull top pixel off stack
   for all fillable pixels A in the span
      fill the pixel
      for neighbor B above and below
         if pixel fillable and A or B at start of span
            push on stack
```

Efficient, low stack demand, few function calls

Variants on flood fill

Fill intervals

Calculate mask (magic wand tool)

Soft fill, smooth edges

Conclusions about low-level algorithms

Not the most common ones to implement - but more common than you may think

Methods often applicable for other problems

Some 2D methods (like the inside-outside test) interesting in 3D generalizations

Text in OpenGL

Text is not OpenGL's task!

Thus, multiple solutions. Which is best?

Possible options

Render to textures on CPU

Outline fonts

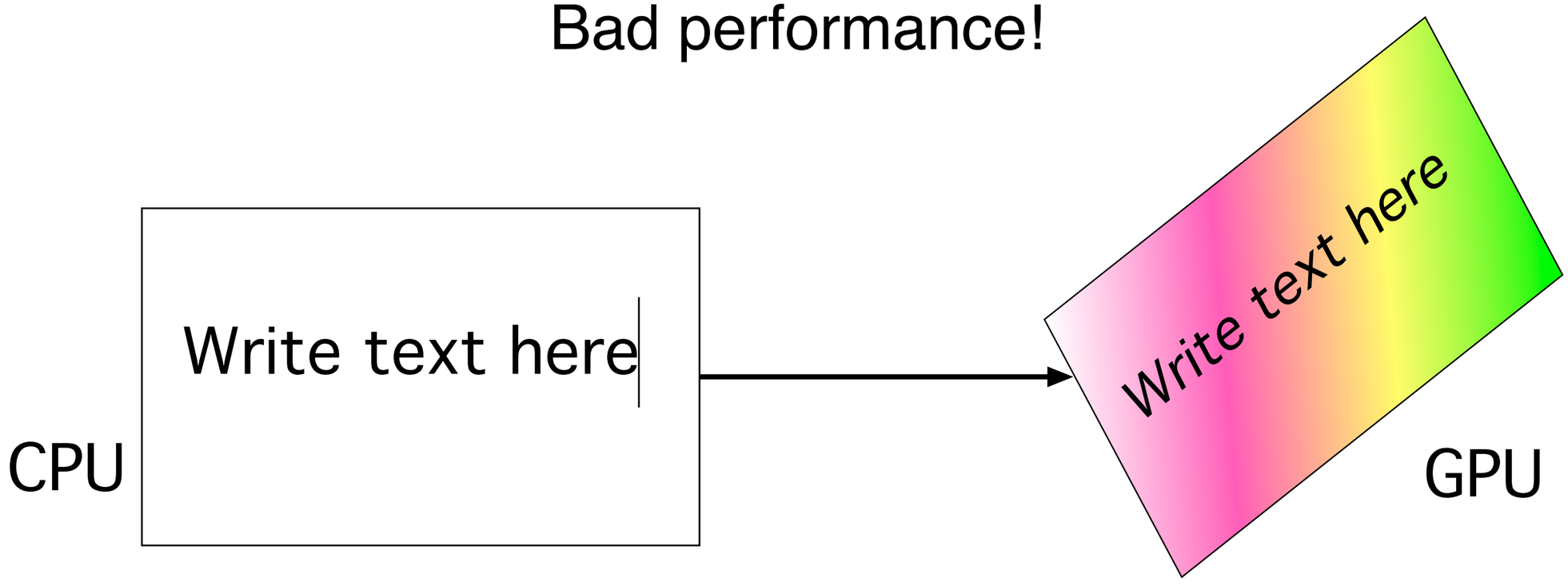
Bitmap fonts

Texture fonts

Render to textures on CPU

Do the rendering on the CPU, with system fonts, with whatever 2D API you have there.

Bad performance!



Outline fonts on GPU

Evaluate splines to create characters.

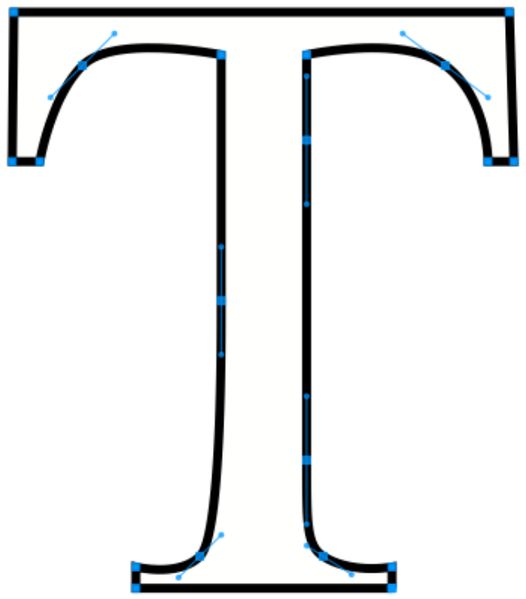
FTGL - FreeType for OpenGL

stb_truetype - single file solution!

Scalable

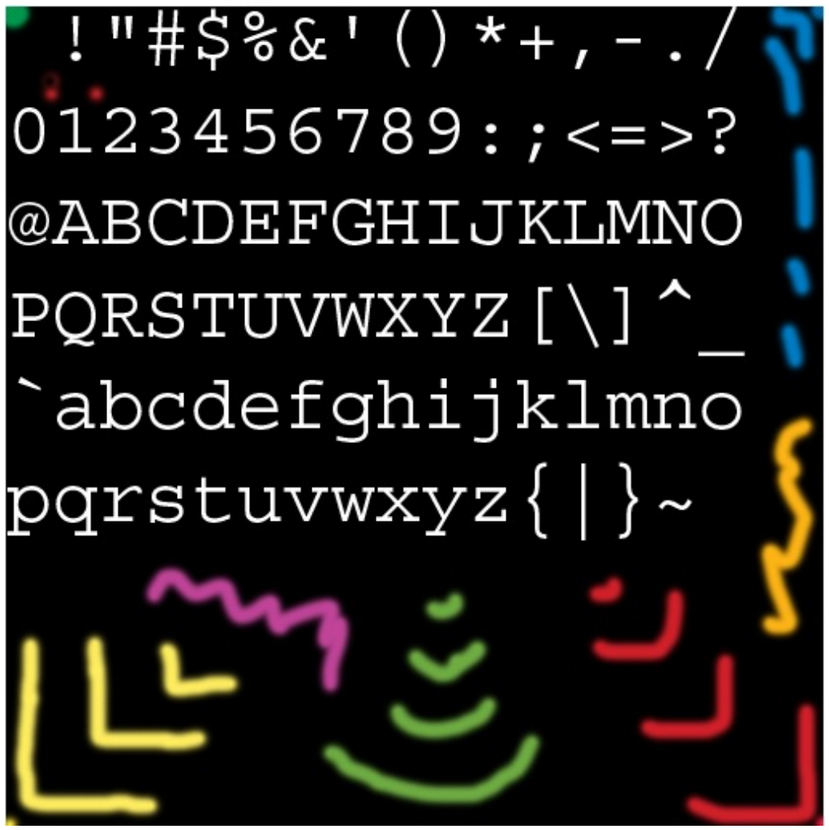
Can create 3D text!

Typically rendered to texture fonts.



Texture fonts

Pre-generated texture with full alphabet

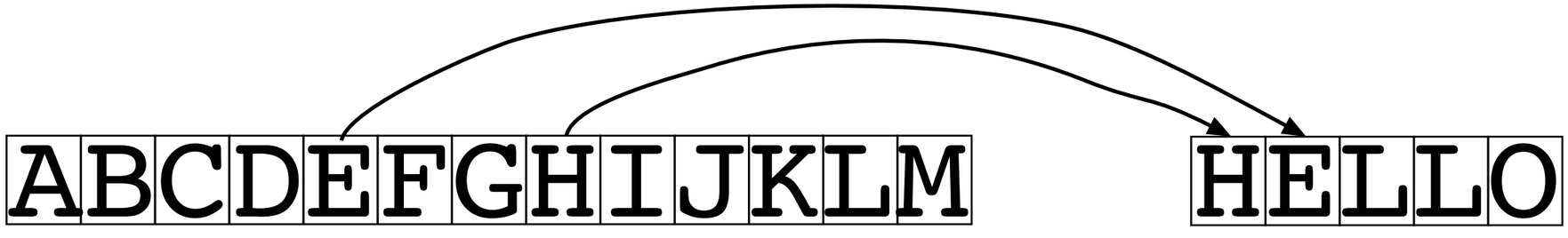


Texture fonts

Assemble parts of the texture from string.

Good performance!

Limited scalability.



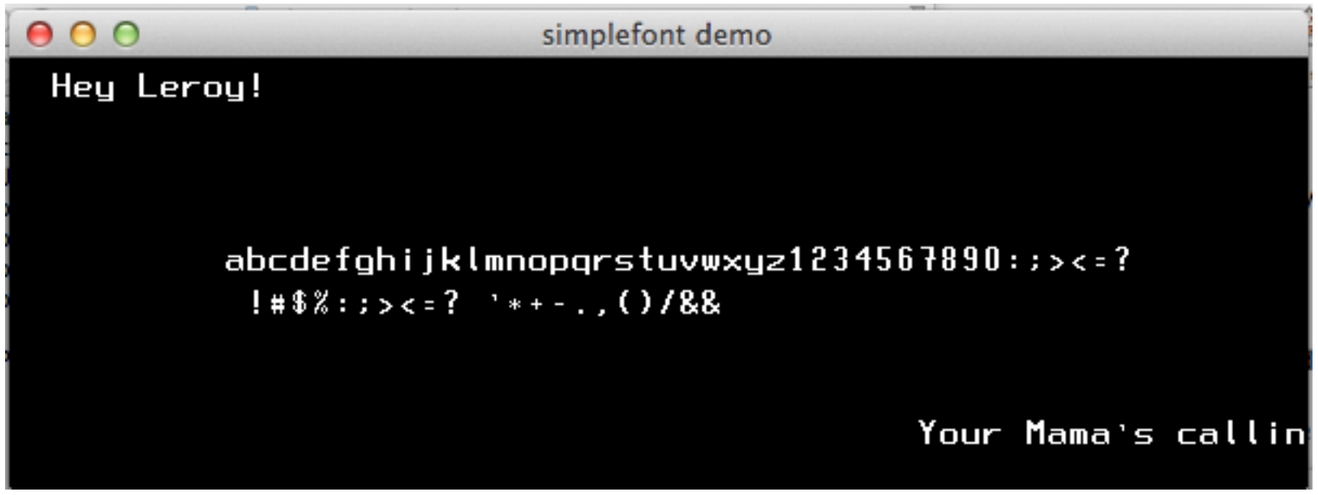
Texture with entire font

Assembled text

SimpleFont - simplified texture font solution

SimpleFont simple unit suitable for project work.

No texture - inline bitmaps.



SimpleFont API

```
GLuint sfMakeRasterFont(void);

void sfDrawString(int h, int v, char *s);

void sfSetRasterSize(int h, int v);
```

Initialize

Draw

Tell it about
window resizing

That's what I call a simple API!

SimpleFont 2 extensions

```
// Extended API for multiple font support and color

GLuint sfLoadExternalFont(char *data, float charWidth, float
charHeight, int imageWidth, int imageHeight, int
extraSpace);

void sfSetFont(GLuint font);

void sfSetFontColor(GLfloat red, GLfloat green, GLfloat blue);
```

```
 !"#$%&'()*+,-./
0123456789:;<=>?
@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_
`abcdefghijklmno
pqrstuvwxyz{|}~
```

```
 !"#$%&'()*+,-./
0123456789:;<=>?
@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_
`abcdefghijklmno
pqrstuvwxyz{|}~
```

A full texture font renderer.
Still not complicated.

stb_truetype and my-stbtt

My adapter of stb_truetype to OpenGL.

Requires TT fonts.

my-stbtt API similar to SimpleFont

Almost as easy as SimpleFont

Very new, might have some quirks.

```
stbfont *my_stbtt_initfont(const char *fontName, float fontsize);

void my_stbtt_init();

float my_stbtt_print(float x, float y, const char *text, stbfont *f);

void my_stbtt_set_color(float r, float g, float b);
```

Playing sounds

Audio is beside the point... but an animation may benefit.

Cross-platform library: OpenAL

Simple API for that: CallMeAL

Only WAV files.

CallMeAl

Small API. Basically Init and Play.

```
int InitCallMeAL(int channels);
void HaltCallMeAL();
ALuint LoadSound(char* filename);
ALint GetChannelStatus(int ch);
char ChannelIsPlaying(int ch);
void PlaySoundInChannel(ALuint buffer, int ch);
void PlayLoopedSoundInChannel(ALuint buffer, int ch);
void PlaySound(ALuint buffer);
void StopSound(int ch);
void StopAllSounds();
```

Spock image removed

a.k.a. glNext

Is the future here? (Released spring 2016)

”Next-generation OpenGL”?

Optional in the labs.

What? When? Where?

Newer API, development started in 2014, released 16th february 2016.

Open specification - don’t expect it to hit all platforms immediately! And it didn't.

Big players try their own solutions. Apple has "Metal" instead. MS works through DirectX.

Background - AMD Mantle

Low-level API for more direct control of the GPU.

Mantle → Vulkan, based on Mantle

”Some features”?

Direct3D

Metal

So what is the point to us?

Promises:

- Lower driver overhead
- More multi-thread friendly than OpenGL
- Shaders can be compiled to intermediary binary format (SPIR-V)
- Open front-end for shader compilers?

Show me some code!

This is how you draw a triangle:

```
#ifndef _MSC_VER
#define _ISOC11_SOURCE /* for aligned_alloc() */
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <assert.h>

#ifdef _WIN32
#pragma comment(linker, "/subsystem:windows")
#define APP_NAME_STR_LEN 80
#endif // _WIN32

#include <vulkan/vulkan.h>

#define DEMO_TEXTURE_COUNT 1
#define VERTEX_BUFFER_BIND_ID 0
#define APP_SHORT_NAME "tri"
#define APP_LONG_NAME "The Vulkan Triangle Demo Program"

#define ARRAY_SIZE(a) (sizeof(a) / sizeof(a[0]))

#if defined(NDEBUG) && defined(__GNUC__)
#define U_ASSERT_ONLY __attribute__((unused))
#else
#define U_ASSERT_ONLY
#endif

#ifdef _WIN32
#define ERR_EXIT(err_msg, err_class)                                           \
    do {                                                                       \
        MessageBox(NULL, err_msg, err_class, MB_OK);                           \
        exit(1);                                                               \
    } while (0)
#else // _WIN32

#define ERR_EXIT(err_msg, err_class)                                           \
    do {                                                                       \
        printf(err_msg);                                                       \
        fflush(stdout);                                                        \
        exit(1);                                                               \
    } while (0)
#endif // _WIN32

#define GET_INSTANCE_PROC_ADDR(inst, entrypoint)                               \
    {                                                                          \
        demo->fp##entrypoint =                                                 \
            (PFN_vk##entrypoint)vkGetInstanceProcAddr(inst, "vk" #entrypoint); \
        if (demo->fp##entrypoint == NULL) {                                    \
            ERR_EXIT("vkGetInstanceProcAddr failed to find vk" #entrypoint,    \
                     "vkGetInstanceProcAddr Failure");                         \
        }                                                                      \
    }

#define GET_DEVICE_PROC_ADDR(dev, entrypoint)                                  \
    {                                                                          \
        demo->fp##entrypoint =                                                 \
            (PFN_vk##entrypoint)vkGetDeviceProcAddr(dev, "vk" #entrypoint);    \
        if (demo->fp##entrypoint == NULL) {                                    \
            ERR_EXIT("vkGetDeviceProcAddr failed to find vk" #entrypoint,      \
                     "vkGetDeviceProcAddr Failure");                           \
        }                                                                      \
    }

struct texture_object {
    VkSampler sampler;

    VkImage image;
    VkImageLayout imageLayout;

    VkDeviceMemory mem;
    VkImageView view;
    int32_t tex_width, tex_height;
};

VKAPI_ATTR VkBool32 VKAPI_CALL
dbgFunc(VkFlags msgFlags, VkDebugReportObjectTypeEXT objType,
        uint64_t srcObject, size_t location, int32_t msgCode,
        const char *pLayerPrefix, const char *pMsg, void *pUserData) {
    char *message = (char *)malloc(strlen(pMsg) + 100);

    assert(message);

    if (msgFlags & VK_DEBUG_REPORT_ERROR_BIT_EXT) {
        sprintf(message, "ERROR: [%s] Code %d : %s", pLayerPrefix, msgCode,
                pMsg);
    } else if (msgFlags & VK_DEBUG_REPORT_WARNING_BIT_EXT) {
        sprintf(message, "WARNING: [%s] Code %d : %s", pLayerPrefix, msgCode,
                pMsg);
    } else {
        return false;
    }

#ifdef _WIN32
    MessageBox(NULL, message, "Alert", MB_OK);
#else
    printf("%s\n", message);
    fflush(stdout);
#endif
    free(message);

    /*
     * false indicates that layer should not bail-out of an
     * API call that had validation failures. This may mean that the
     * app dies inside the driver due to invalid parameter(s).
     * That's what would happen without validation layers, so we'll
     * keep that behavior here.
     */
    return false;
}

typedef struct _SwapchainBuffers {
    VkImage image;
    VkCommandBuffer cmd;
    VkImageView view;
} SwapchainBuffers;

struct demo {
#ifdef _WIN32
#define APP_NAME_STR_LEN 80
    HINSTANCE connection;        // hInstance - Windows Instance
    char name[APP_NAME_STR_LEN]; // Name to put on the window/icon
    HWND window;                 // hWnd - window handle
#else                            // _WIN32
    xcb_connection_t *connection;
    xcb_screen_t *screen;
    xcb_window_t window;
    xcb_intern_atom_reply_t *atom_wm_delete_window;
#endif                           // _WIN32
    VkSurfaceKHR surface;
    bool prepared;
    bool use_staging_buffer;

    VkInstance inst;
    VkPhysicalDevice gpu;
    VkDevice device;
    VkQueue queue;
    VkPhysicalDeviceProperties gpu_props;
    VkQueueFamilyProperties *queue_props;
    uint32_t graphics_queue_node_index;

    uint32_t enabled_extension_count;
    uint32_t enabled_layer_count;
    char *extension_names[64];
    char *device_validation_layers[64];

    int width, height;
    VkFormat format;
    VkColorSpaceKHR color_space;

    PFN_vkGetPhysicalDeviceSurfaceSupportKHR
        fpGetPhysicalDeviceSurfaceSupportKHR;
    PFN_vkGetPhysicalDeviceSurfaceCapabilitiesKHR
        fpGetPhysicalDeviceSurfaceCapabilitiesKHR;
    PFN_vkGetPhysicalDeviceSurfaceFormatsKHR
        fpGetPhysicalDeviceSurfaceFormatsKHR;
    PFN_vkGetPhysicalDeviceSurfacePresentModesKHR
        fpGetPhysicalDeviceSurfacePresentModesKHR;
    PFN_vkCreateSwapchainKHR fpCreateSwapchainKHR;
    PFN_vkDestroySwapchainKHR fpDestroySwapchainKHR;
    PFN_vkGetSwapchainImagesKHR fpGetSwapchainImagesKHR;
    PFN_vkAcquireNextImageKHR fpAcquireNextImageKHR;
    PFN_vkQueuePresentKHR fpQueuePresentKHR;
    uint32_t swapchainImageCount;
    VkSwapchainKHR swapchain;
    SwapchainBuffers *buffers;

    VkCommandPool cmd_pool;

    struct {
        VkFormat format;

        VkImage image;
        VkDeviceMemory mem;
        VkImageView view;
    } depth;

    struct texture_object textures[DEMO_TEXTURE_COUNT];

    struct {
        VkBuffer buf;
        VkDeviceMemory mem;

        VkPipelineVertexInputStateCreateInfo vi;
        VkVertexInputBindingDescription vi_bindings[1];
        VkVertexInputAttributeDescription vi_attrs[2];
    } vertices;

    VkCommandBuffer setup_cmd; // Command Buffer for initialization commands
    VkCommandBuffer draw_cmd;  // Command Buffer for drawing commands
    VkPipelineLayout pipeline_layout;
    VkDescriptorSetLayout desc_layout;
    VkPipelineCache pipelineCache;
    VkRenderPass render_pass;
    VkPipeline pipeline;

    VkShaderModule vert_shader_module;
    VkShaderModule frag_shader_module;

    VkDescriptorPool desc_pool;
    VkDescriptorSet desc_set;

    VkFramebuffer *framebuffers;

    VkPhysicalDeviceMemoryProperties memory_properties;

    bool validate;
    PFN_vkCreateDebugReportCallbackEXT CreateDebugReportCallback;
    PFN_vkDestroyDebugReportCallbackEXT DestroyDebugReportCallback;
    VkDebugReportCallbackEXT msg_callback;
    PFN_vkDebugReportMessageEXT DebugReportMessage;

    float depthStencil;
    float depthIncrement;

    bool quit;
    uint32_t current_buffer;
    uint32_t queue_count;
};

// Forward declaration:
static void demo_resize(struct demo *demo);

static bool memory_type_from_properties(struct demo *demo, uint32_t typeBits,
                                        VkFlags requirements_mask,
                                        uint32_t *typeIndex) {
    // Search memtypes to find first index with those properties
    for (uint32_t i = 0; i < 32; i++) {
        if ((typeBits & 1) == 1) {
            // Type is available, does it match user properties?
            if ((demo->memory_properties.memoryTypes[i].propertyFlags &
                 requirements_mask) == requirements_mask) {
                *typeIndex = i;
                return true;
            }
        }
        typeBits >>= 1;
    }
    // No memory types matched, return failure
    return false;
}

static void demo_flush_init_cmd(struct demo *demo) {
    VkResult U_ASSERT_ONLY err;

    if (demo->setup_cmd == VK_NULL_HANDLE)
        return;

    err = vkEndCommandBuffer(demo->setup_cmd);
    assert(!err);

    const VkCommandBuffer cmd_bufs[] = {demo->setup_cmd};
    VkFence nullFence = {VK_NULL_HANDLE};
    VkSubmitInfo submit_info = {.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO,
                                .pNext = NULL,
                                .waitSemaphoreCount = 0,
                                .pWaitSemaphores = NULL,
                                .pWaitDstStageMask = NULL,
                                .commandBufferCount = 1,
                                .pCommandBuffers = cmd_bufs,
                                .signalSemaphoreCount = 0,
                                .pSignalSemaphores = NULL};

    err = vkQueueSubmit(demo->queue, 1, &submit_info, nullFence);
    assert(!err);

    err = vkQueueWaitIdle(demo->queue);
    assert(!err);

    vkFreeCommandBuffers(demo->device, demo->cmd_pool, 1, cmd_bufs);
    demo->setup_cmd = VK_NULL_HANDLE;
}

static void demo_set_image_layout(struct demo *demo, VkImage image,
                                  VkImageAspectFlags aspectMask,
                                  VkImageLayout old_image_layout,
                                  VkImageLayout new_image_layout,
                                  VkAccessFlagBits srcAccessMask) {

    VkResult U_ASSERT_ONLY err;
```

These are the first ≠300 lines of a file with 2448 lines... should I continue?

What kind of mind-boggling bullsh*t is this?

- Multi-threaded
- Low-level
- Detailed control over buffers, processes, queues, devices...

This comes for a cost!

Important: This does NOT mean that OpenGL will be discontinued in the foreseeable future! Vulkan is a lower-level complement, not a replacement!

An alternative, that (not only) we are exploring: Simplified APIs on Vulkan. Some of you have been using it in the labs.

Much is happening! Computer graphics is still moving fast...

Problem for Vulkan: The big players!

MicroSoft packages everything in DirectX, but Vulkan is available for Windows.

Apple also made their own, "Metal" But "Molten" maps Metal to the Vulkan API as well as GLES.

Apple and Microsoft are also the worst at supporting OpenGL!

Will "lock-in" strategies win or the open standards?

WebGL and WebGPU

Graphics for web browsers!

WebGL established.

WebGPU more in progress.

On WebGL: Performance

CPU part, not so good. Limited by JavaScript. But JavaScript is faster than you might expect!

GPU part runs on the GPU as usual - can be very good!

More important than ever to put workload on the GPU!

Problems

WebGL: Mostly everything outside OpenGL! JavaScript add-ons for loading files, user input. OpenGL mostly as usual.

WebGPU: Promises support for several languages. (What will this mean to us?)

WebGPU

Basically Vulkan for the web browser.

Newer technology, under development. (Available in most browsers as late as 2025.)

Lower level than WebGL, potentially more efficient.

Vulkan and WebGPU

Much is happening right now!

Helper libraries, tools, pre-processors... accessability for teaching? Many open questions.

Will you be among those who answers them?

Next time

Randomness

Noise

Fractals based on noise

FBM

Diffusion