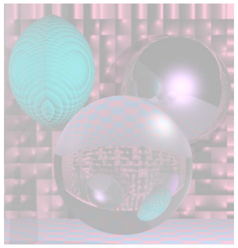
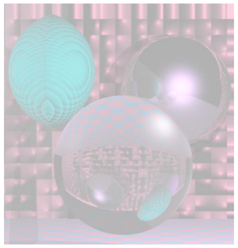


CS G140
Graduate Computer Graphics

Prof. Harriet Fell

Spring 2009

Lecture 7 – February 18, 2009



Today’s Topics

- Poly Mesh
  - Hidden Surface Removal
  - Visible Surface Determination

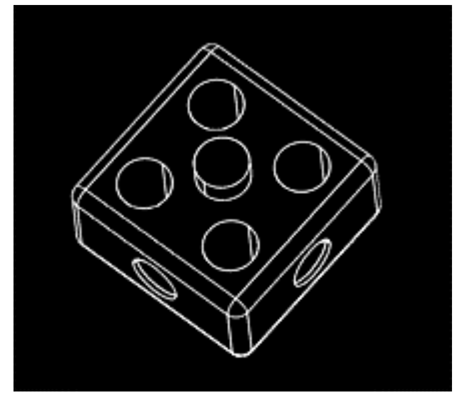
- Noise and Turbulence
  - Clouds
  - Marble
  - Other Effects

Rendering a Polymesh

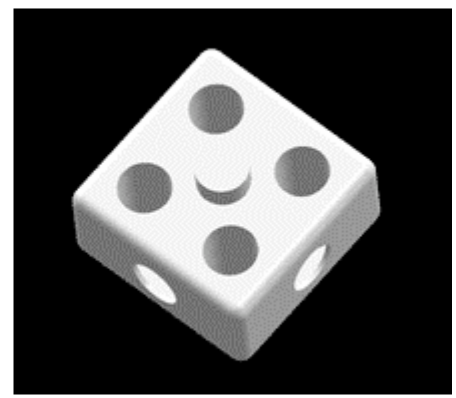
- Scene is composed of triangles or other polygons.
- We want to view the scene from different view-points.
  - Hidden Surface Removal
    - Cull out surfaces or parts of surfaces that are not visible.
  - Visible Surface Determination
    - Head right for the surfaces that are visible.
    - Ray-Tracing is one way to do this.

Wireframe Rendering

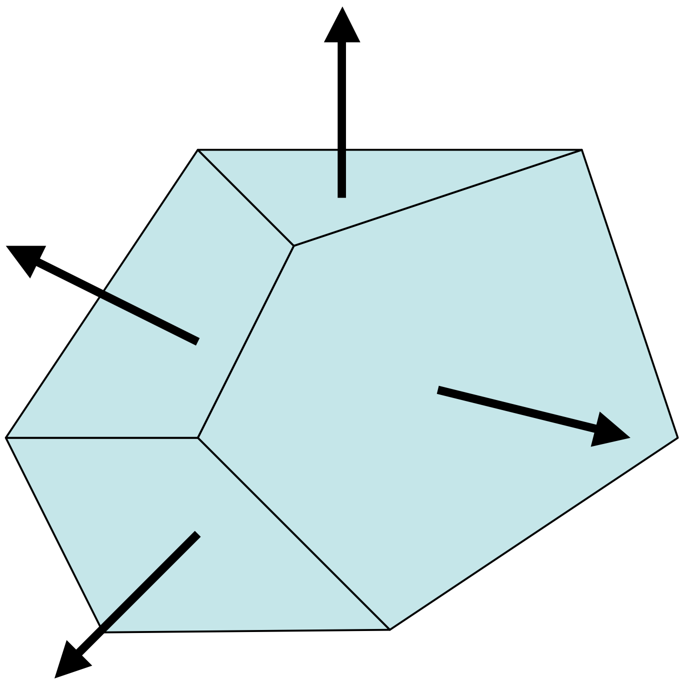
Hidden-Line Removal



Hidden-Face Removal



Convex Polyhedra



We can see a face if and only if its normal has a component toward us.

$$\boldsymbol{N} \cdot \boldsymbol{V} > 0$$

V points from the face toward the viewer.

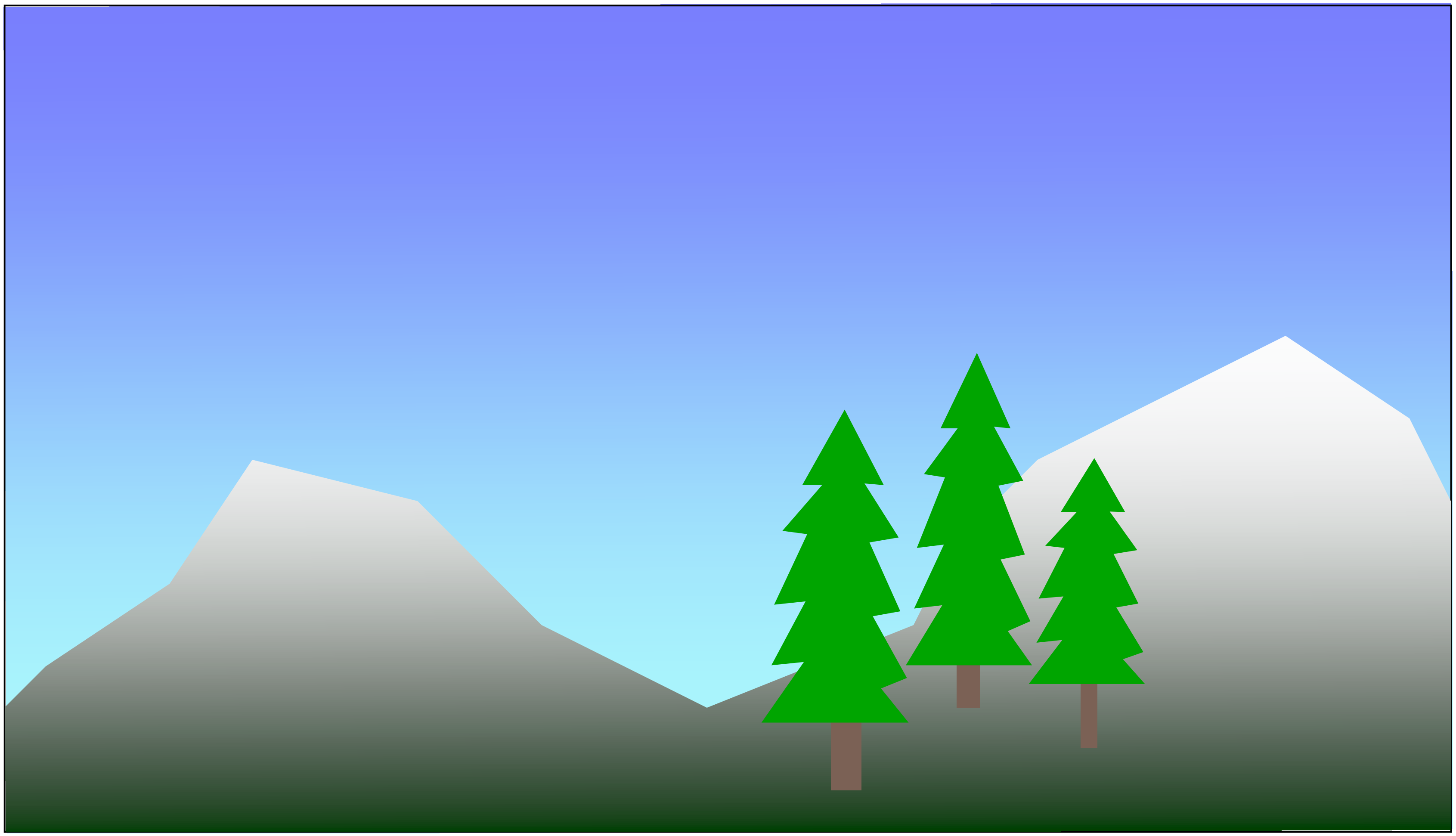
N point toward the outside of the polyhedra.

Hidden Surface Removal

- Backface culling
  - Never show the back of a polygon.
- Viewing frustum culling
  - Discard objects outside the camera's view.
- Occlusion culling
  - Determining when portions of objects are hidden.
    - Painter's Algorithm
    - Z-Buffer
- Contribution culling
  - Discard objects that are too far away to be seen.

http://en.wikipedia.org/wiki/Hidden_face_removal

Painter’s Algorithm



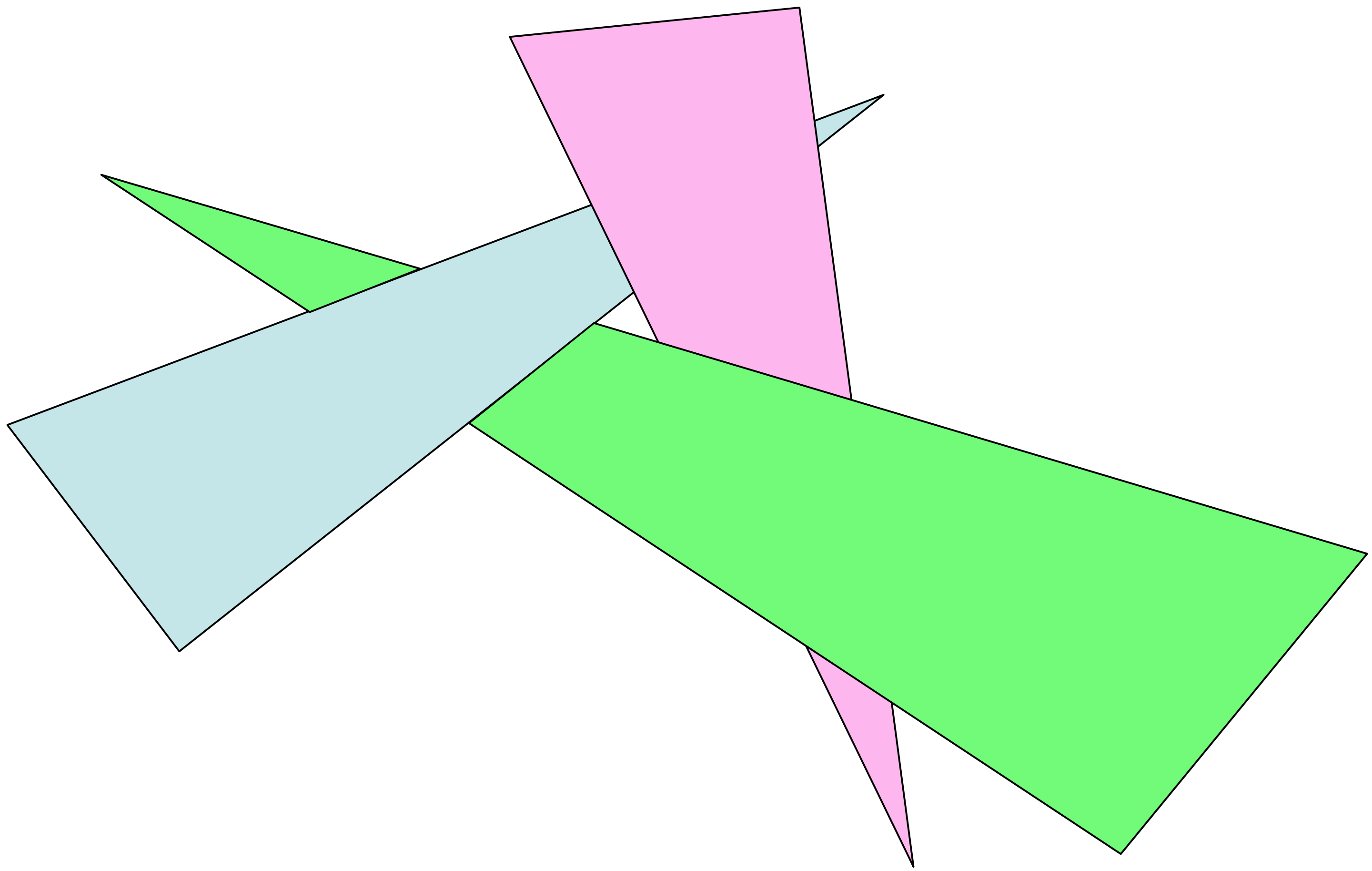
Painter’s Algorithm

Sort objects back to front relative to the
viewpoint.

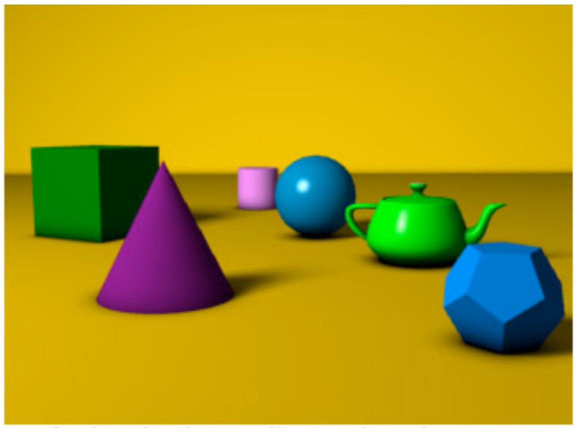
for each object (in the above order) **do**

draw it on the screen

Painter's Problem



Z-Buffer


A simple three dimensional scene


Z-buffer representation

The ***Z-Buffer*** is usually part of graphics card hardware. It can also be implemented in software. The ***Z-Buffer*** is a 2D array that holds one value for each pixel.

The depth of each pixel is stored in the z-buffer.

An object is rendered at a pixel only if its z-value is higher(lower) than the buffer value. The buffer is then updated.

Visible Surface Determination

- If surfaces are invisible, don't render them.
  - Ray Tracing
    - We only render the nearest object.
  - Binary Space Partitioning (BSP)
    - Recursively cut up space into convex sets with hyperplanes.
    - The scene is represented by a BSP-tree.

Sorting the Polygons

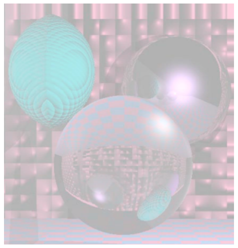
The first step of the Painter's algorithm is:

Sort objects back to front relative to the viewpoint.

The relative order may not be well defined.

We have to reorder the objects when we change the viewpoint.

The BSP algorithm and BSP trees solve these problems.



Binary Space Partition

- Our scene is made of triangles.
  - Other polygons can work too.
- Assume no triangle crosses the plane of any other triangle.
  - We relax this condition later.

following Shirley *et al.*

BSP – Basics

- Let a plane in 3-space (or line in 2-space) be defined implicitly, i.e.
  - $f(\boldsymbol{P}) = f(x, y, z) = 0$ in 3-space
  - $f(\boldsymbol{P}) = f(x, y) = 0$ in 2-space
- All the points $\boldsymbol{P}$ such that $f(\boldsymbol{P}) > 0$ lie on one side of the plane (line).
- All the points $\boldsymbol{P}$ such that $f(\boldsymbol{P}) < 0$ lie on the other side of the plane (line).
- Since we have assumed that all vertices of a triangle lie on the same side of the plane (line), we can tell which side of a plane a triangle lies on.

BSP on a Simple Scene

Suppose scene has 2 triangles

$T1$ on the plane $f(\boldsymbol{P}) = 0$

$T2$ on the $f(\boldsymbol{P}) < 0$ side

e is the eye.

if $f(e) < 0$ **then**

draw $T1$; draw $T2$

else

draw $T2$; draw $T1$

The BSP Tree

Suppose scene has many triangles, *T1, T2, ...* .

We still assume no triangle crosses the plane of any other triangle.

Let $f_i(\boldsymbol{P}) = 0$ be the equation of the plane containing *Ti.*

The *BSPTREE* has a node for each triangle with *T1* at the root.

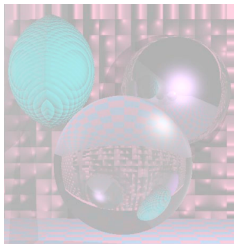
At the node for *Ti,*

the minus subtree contains all the triangles whose vertices have $f_i(\boldsymbol{P}) < 0$

the plus subtree contains all the triangles whose vertices have $f_i(\boldsymbol{P}) > 0$.

BSP on a non-Simple Scene

function draw(bsptree tree, point *e*)
if (tree.empty) **then**
    return
if ($f_{tree.root}(e) < 0$) **then**
    draw(tree.plus, *e*)
    render tree.triangle
    draw(tree.minus, *e*)
else
    draw(tree.minus, *e*)
    render tree.triangle
    draw(tree.plus, *e*)



2D BSP Trees Demo

http://symbolcraft.com/graphics/bsp/

This is a demo in 2 dimensions.

The objects are line segments.

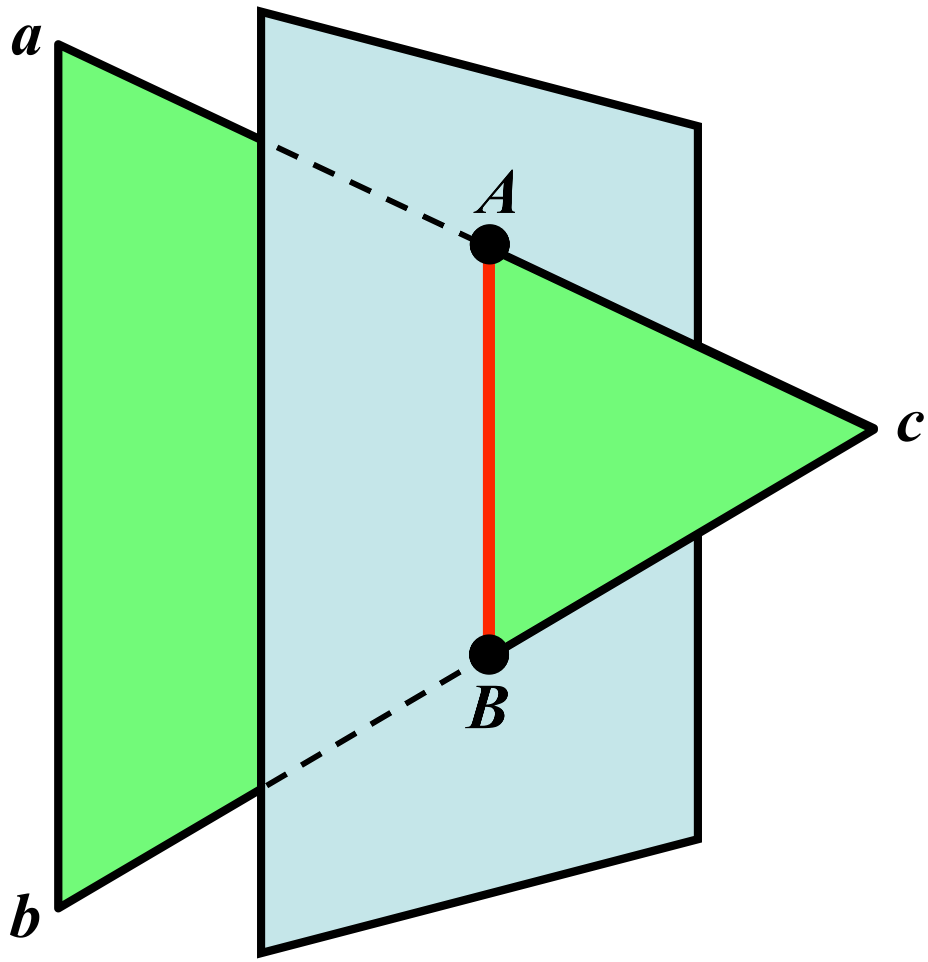
The dividing hyperplanes are lines.

Building the BSP Tree

We still assume no triangle crosses the plane of another triangle.

```
tree = node(T1)
for  i ∈ {2, …, N} do tree.add(Ti)

function add (triangle T)
if (f(a) < 0 and f(b) < 0 and f(c) < 0) then
   if (tree.minus.empty) then
      tree.minus = node(T)
   else
      tree.minus.add(T)
else if (f(a) > 0 and f(b) > 0 and f(c) > 0) then
   if (tree.plus.empty) then
      tree.plus = node(T)
   else
      tree.plus.add(T)
```

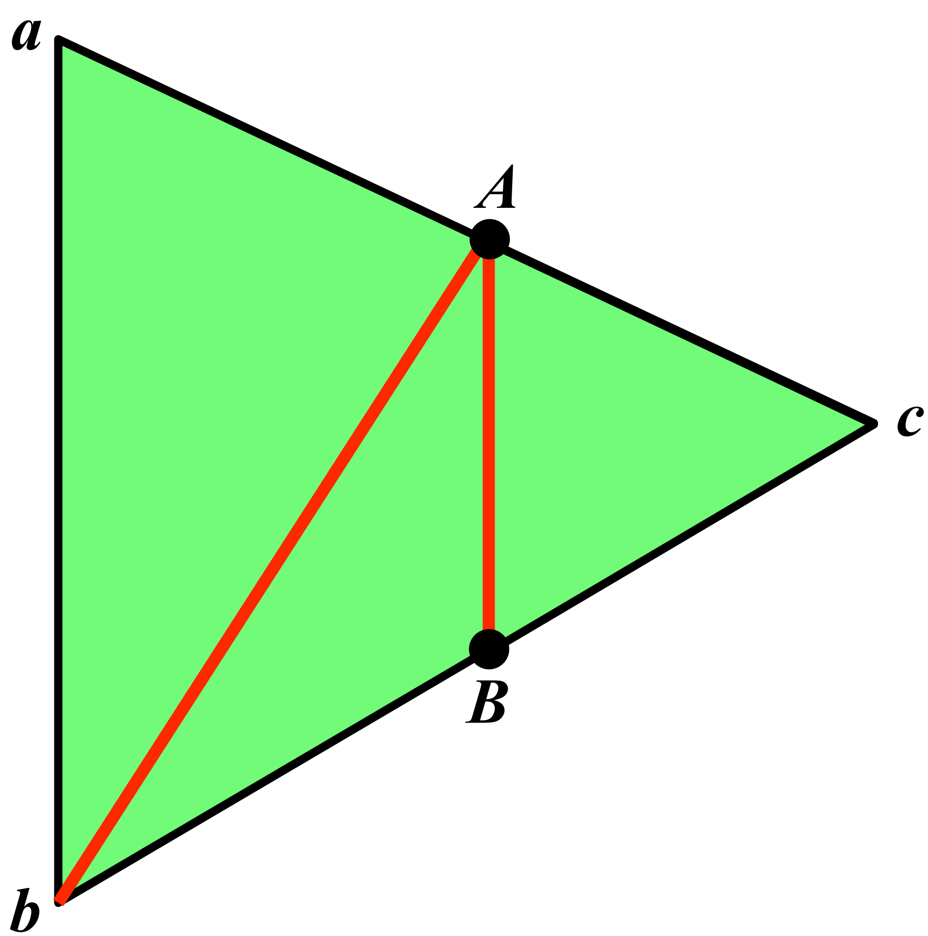
Triangle Crossing a Plane



Two vertices, ***a*** and ***b***, will be on one side and one, ***c***, on the other side.

Find intercepts , ***A*** and ***B***, of the plane with the 2 edges that cross it.

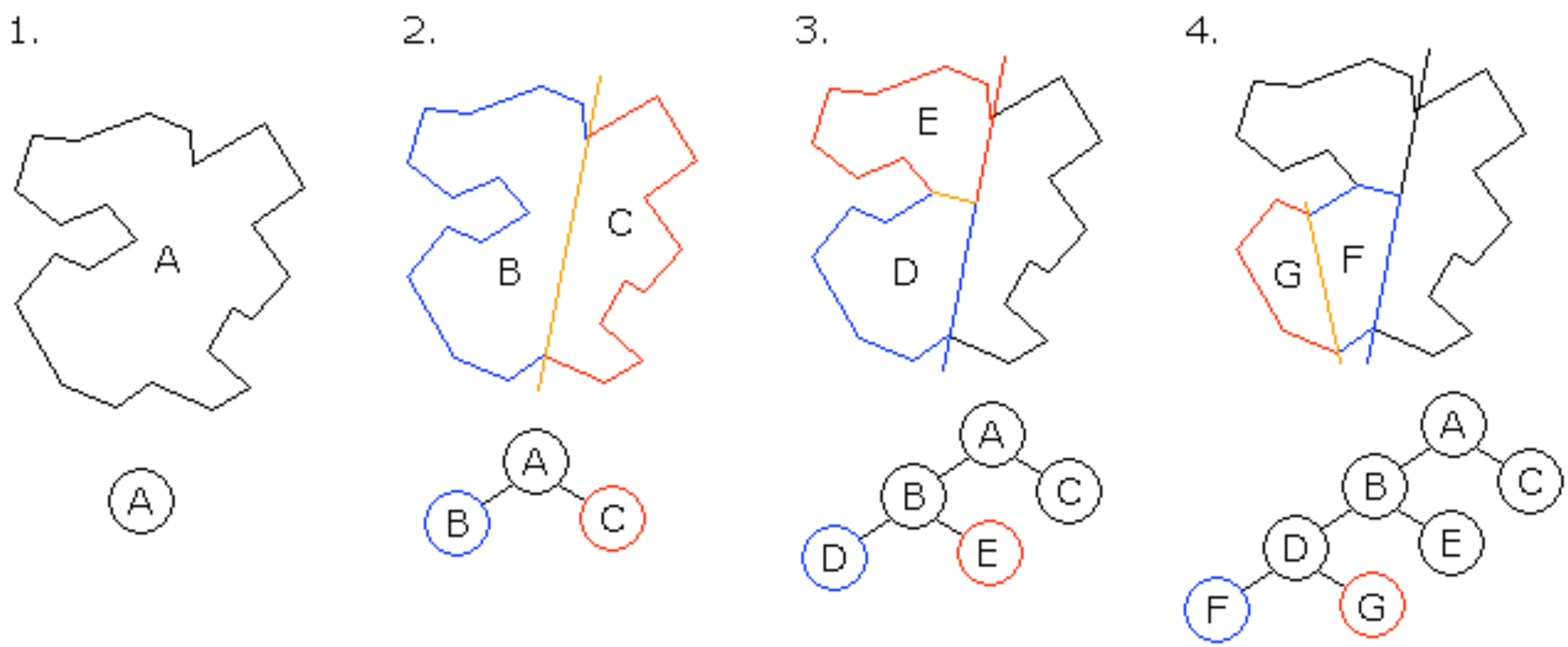
Cutting the Triangle



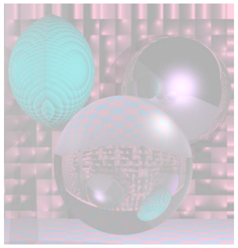
Cut the triangle into three triangles, none of which cross the cutting plane.

Be careful when one or more of a, b, and c is close to or on the cutting plane.

Binary Space Partition of Polygons



by Fredrik (public domain)
http://en.wikipedia.org/wiki/User:Fredrik

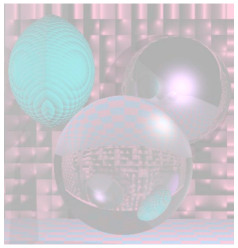


Scan-Line Algorithm

- Romney, G. W., G. S. Watkins, D. C. Evans, "Real-Time Display of Computer Generated Half-Tone Perspective Pictures", *IFIP*, 1968, 973-978.

- **Scan Line Conversion of Polymesh -** like Polyfill

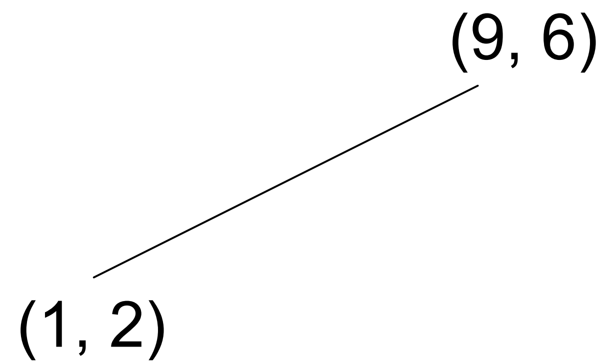
- **Edge Coherence / Scanline Coherence**
- **1)** Most edges don't hit a given scanline- keep track of those that do.
- **2)** Use the last point on an edge to compute the next one. $x_{i+1} = x_i + 1/m$



Polygon Data Structure

edges

xmin	ymax	1/m	• →
1	6	8/4	• →



xmin = x value at lowest y

ymax = highest y

Why 1/m?

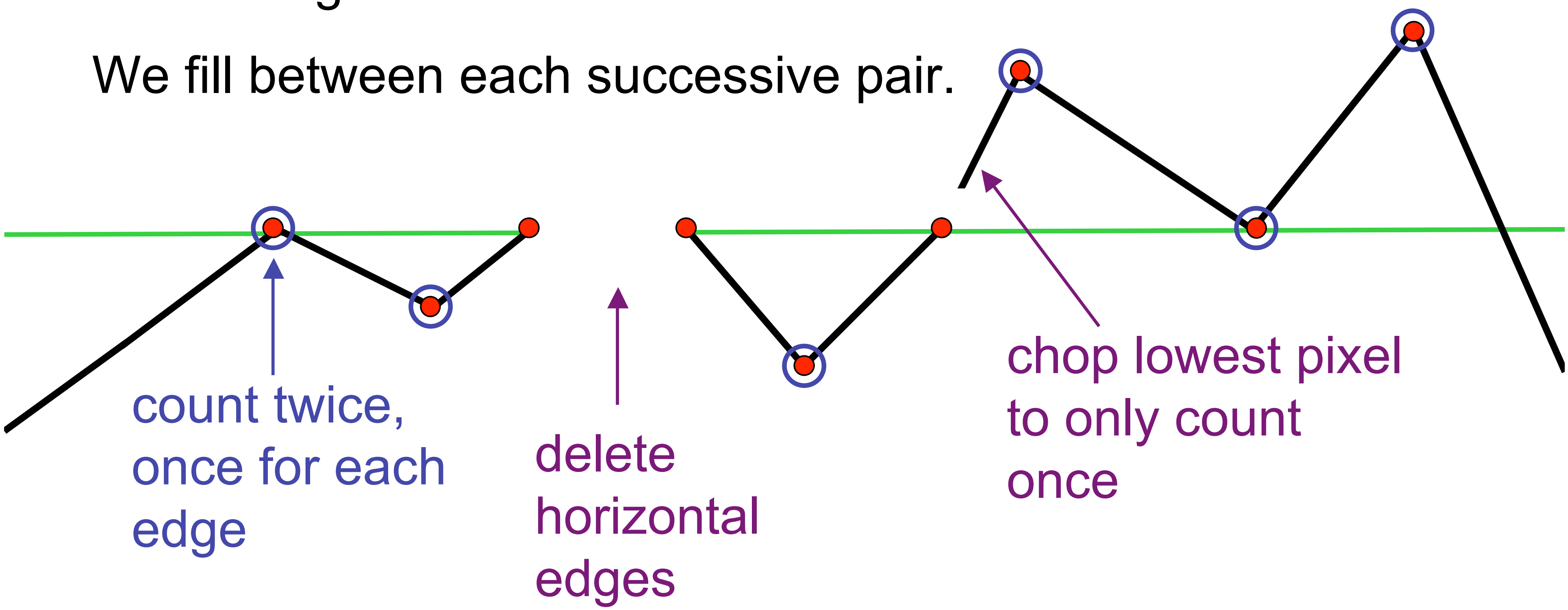
If $y = mx + b$, $x = (y-b)/m$.

x at $y+1 = (y+1-b)/m = (y-b)/m + 1/m$.

Preprocessing the edges

For a closed polygon, there should be an even number of crossings at each scan line.

We fill between each successive pair.



Polygon Data Structure after preprocessing

Edge Table (ET) has a list of edges for each scan line.

13

12

11 $\rightarrow$ e6

10

9

8

7 $\rightarrow$ e3 $\rightarrow$ e4 $\rightarrow$ e5

6 $\rightarrow$ e7 $\rightarrow$ e8

5

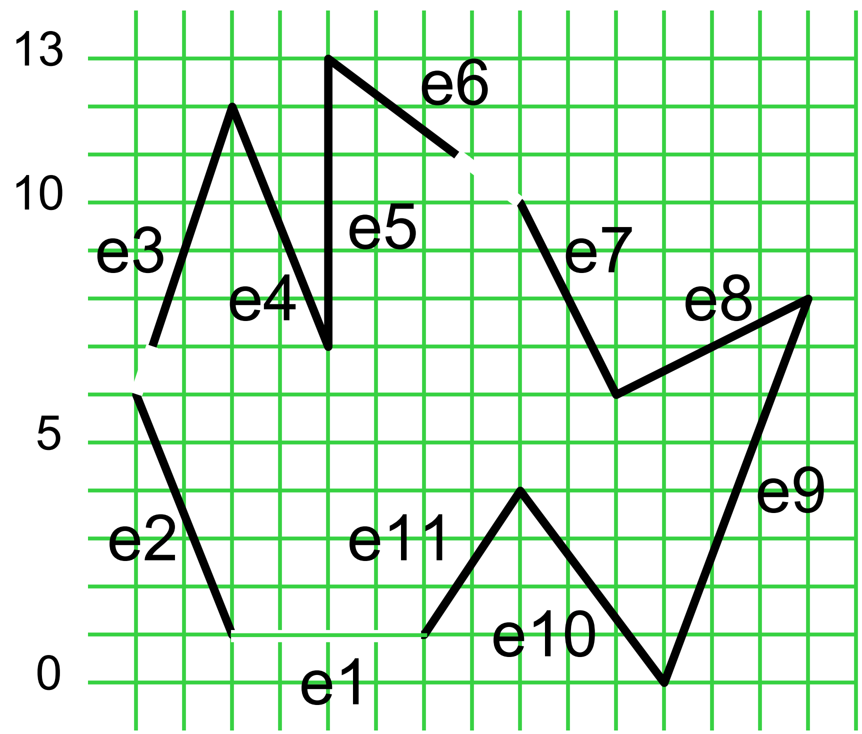
4

3

2

1 $\rightarrow$ e2 $\rightarrow$ e11

0 $\rightarrow$ e10 $\rightarrow$ e9



The Algorithm

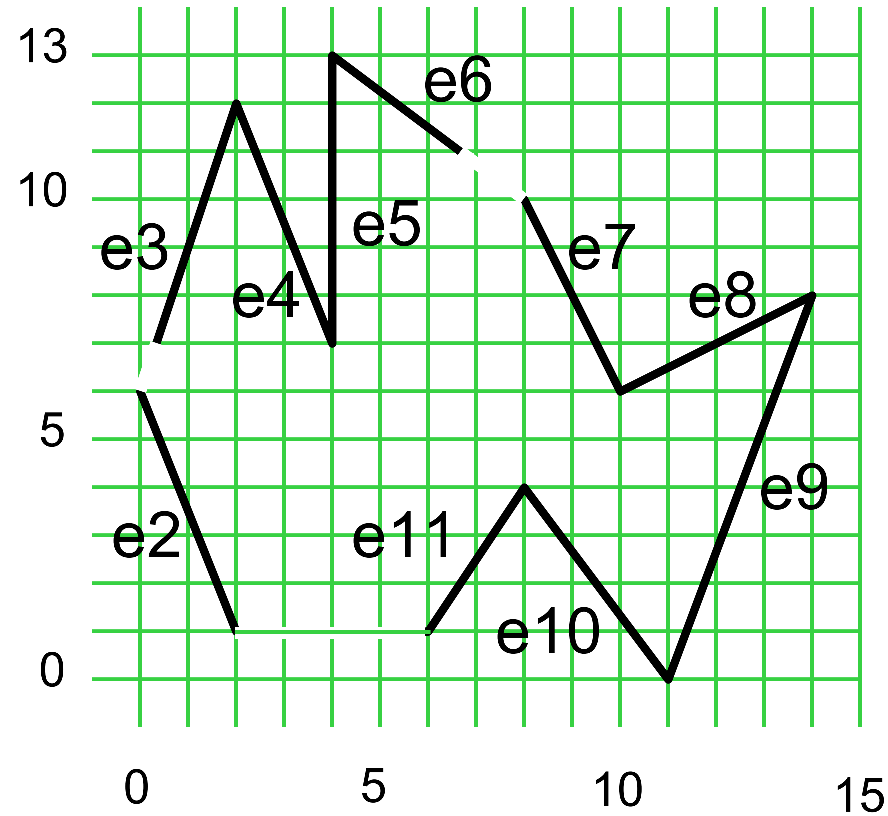
1. Start with smallest nonempty y value in ET.
2. Initialize SLB (Scan Line Bucket) to *nil*.
3. While current y ≤ top y value:
   a. Merge y bucket from ET into SLB; sort on xmin.
   b. Fill pixels between rounded pairs of x values in SLB.
   c. Remove edges from SLB whose ytop = current y.
   d. Increment xmin by 1/m for edges in SLB.
   e. Increment y by 1.

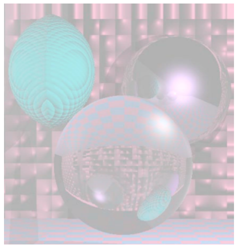
Running the Algorithm

ET

13
12
11 → e6
10
9
8
7 → e3 → e4 → e5
6 → e7 → e8
5
4
3
2
1 → e2 → e11
0 → e10 → e9

	xmin	ymax	1/m
e2	2	6	-2/5
e3	1/3	12	1/3
e4	4	12	-2/5
e5	4	13	0
e6	6 2/3	13	-4/3
e7	10	10	-1/2
e8	10	8	2
e9	11	8	3/8
e10	11	4	-3/4
e11	6	4	2/3





Running the Algorithm

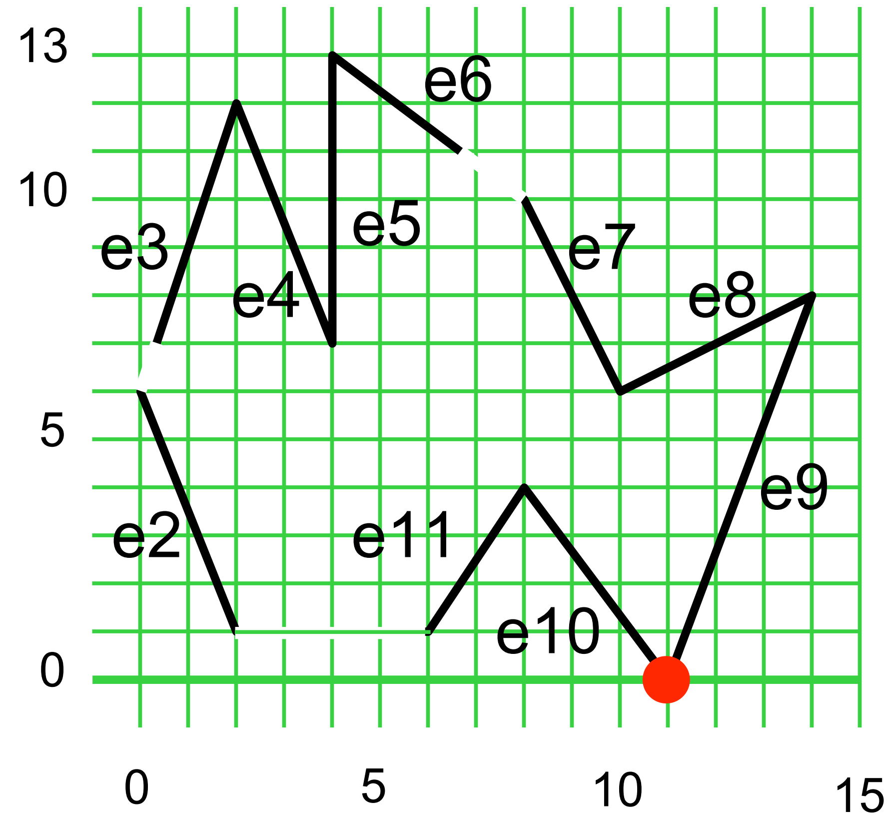
y=0

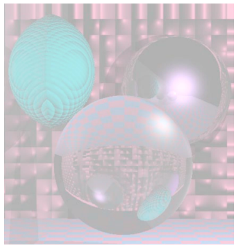
e10

SCB→ | 10 1/4 | 4 | -3/4 | • |

e9

| 11 3/8 | 8 | 3/8 | • |





Running the Algorithm

y=1

SLB→

e2

1 3/5	6	-2/5	•

e11

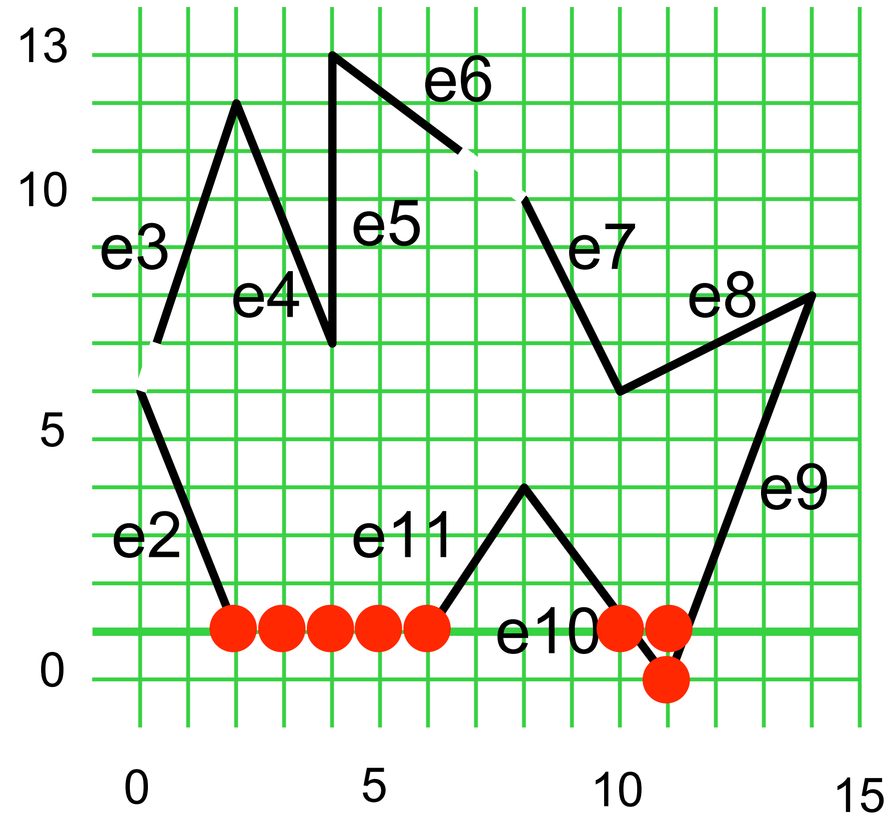
6 2/3	4	2/3	•

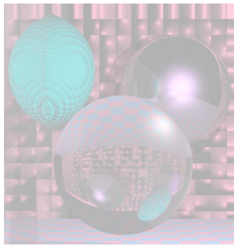
e10

9 1/2	4	-3/4	•

e9

11 6/8	8	3/8	•





Running the Algorithm

y=2

SLB→

e2

1 1/5	6	-2/5	•

e11

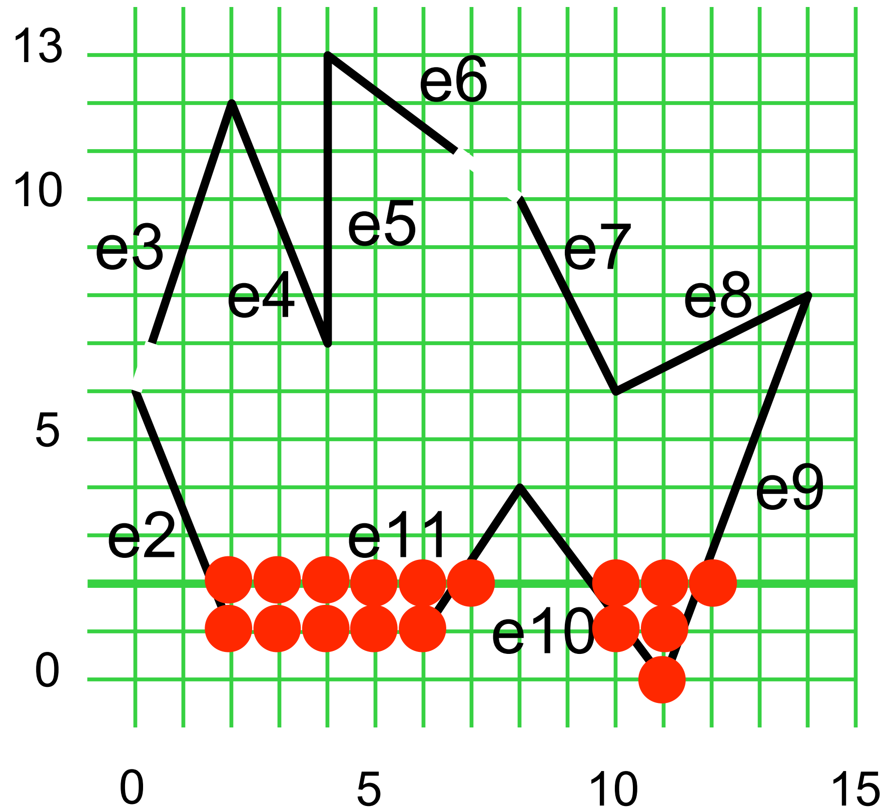
7 1/3	4	2/3	•

e10

8 3/4	4	-3/4	•

e9

12 1/8	8	3/8	•



Running the Algorithm

y=3

SLB→

e2

4/5	6	-2/5	•

e11

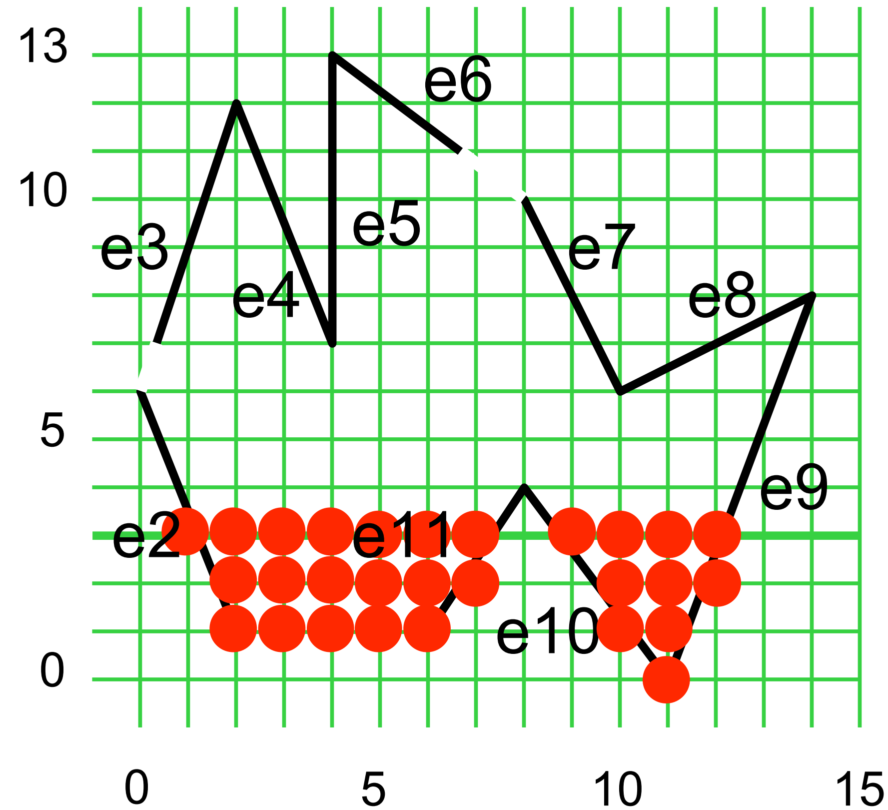
8	4	2/3	•

e10

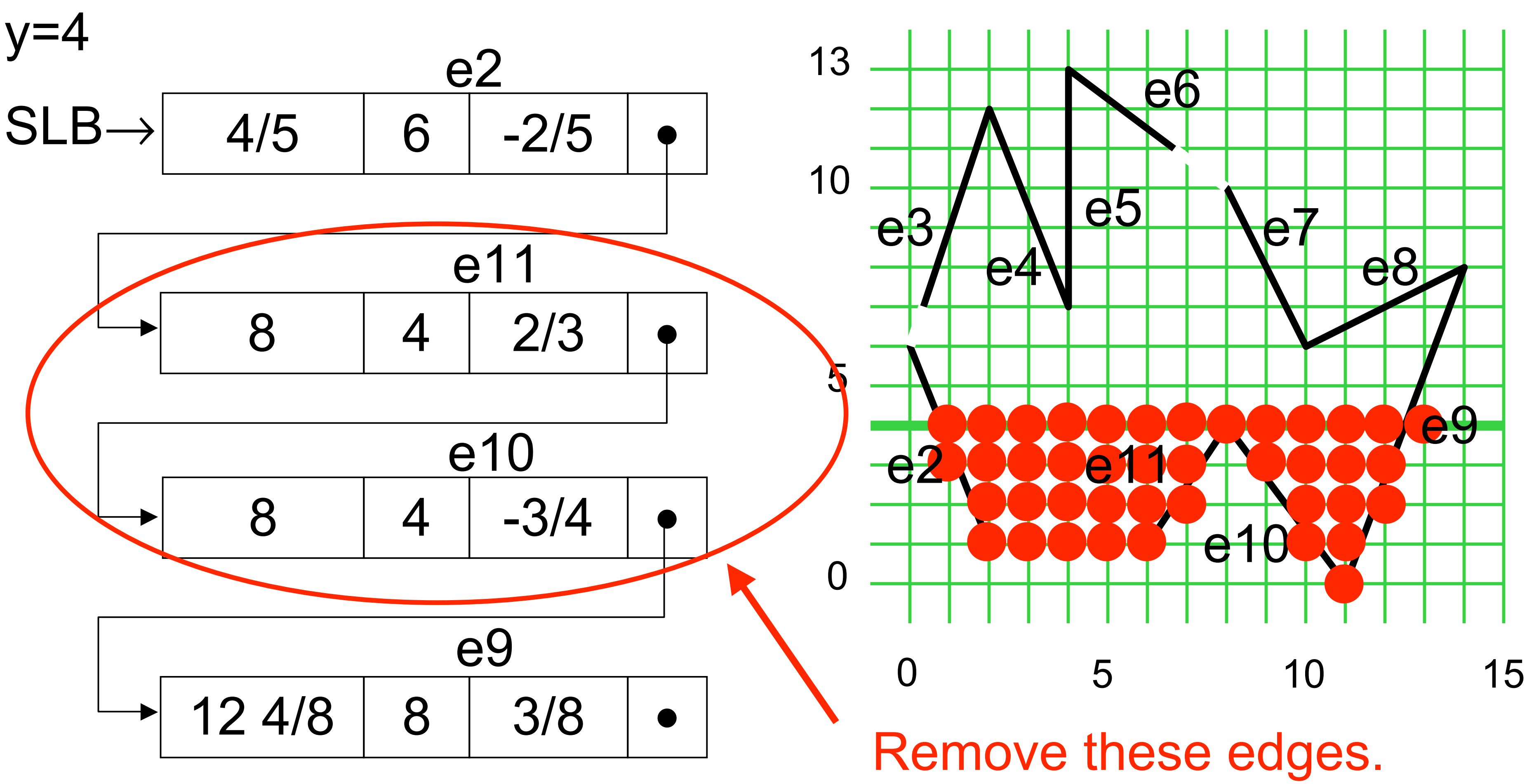
8	4	-3/4	•

e9

12 4/8	8	3/8	•



Running the Algorithm



Running the Algorithm

y=4

SLB→

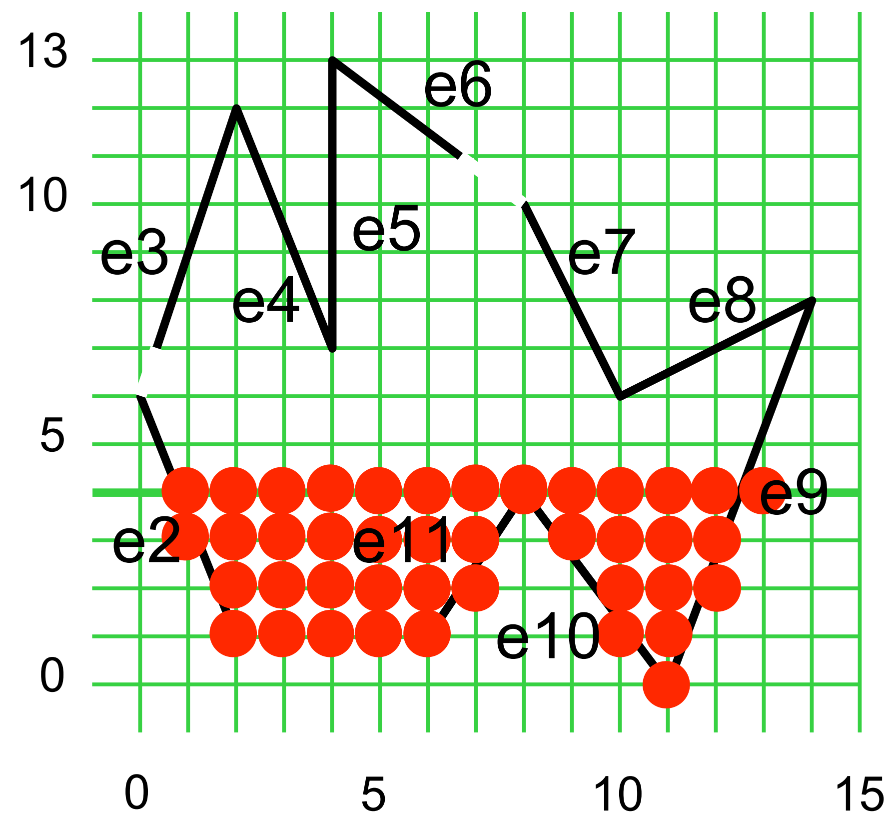
e2

2/5	6	-2/5	•

e9

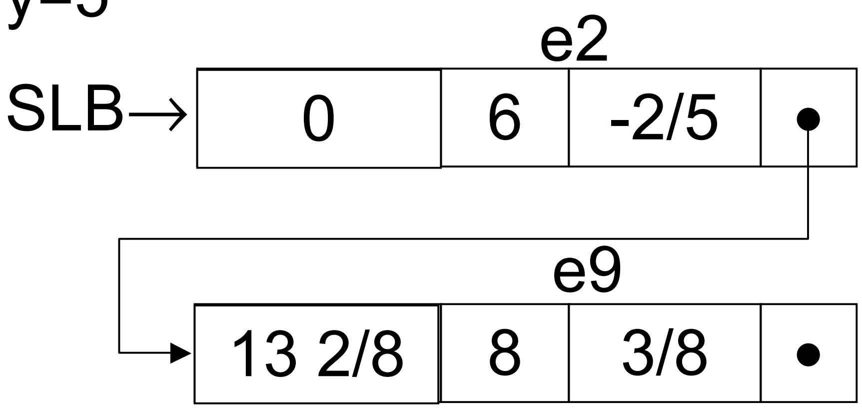
12 7/8	8	3/8	•

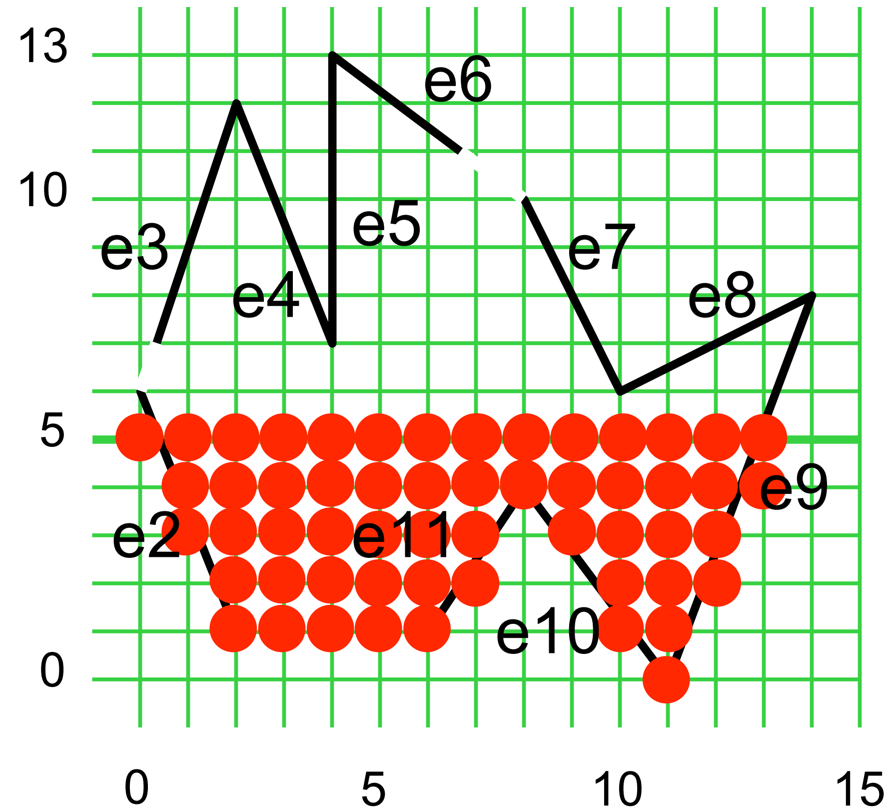
e11 and e10 are removed.



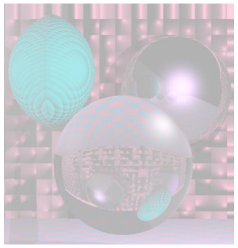
Running the Algorithm

y=5

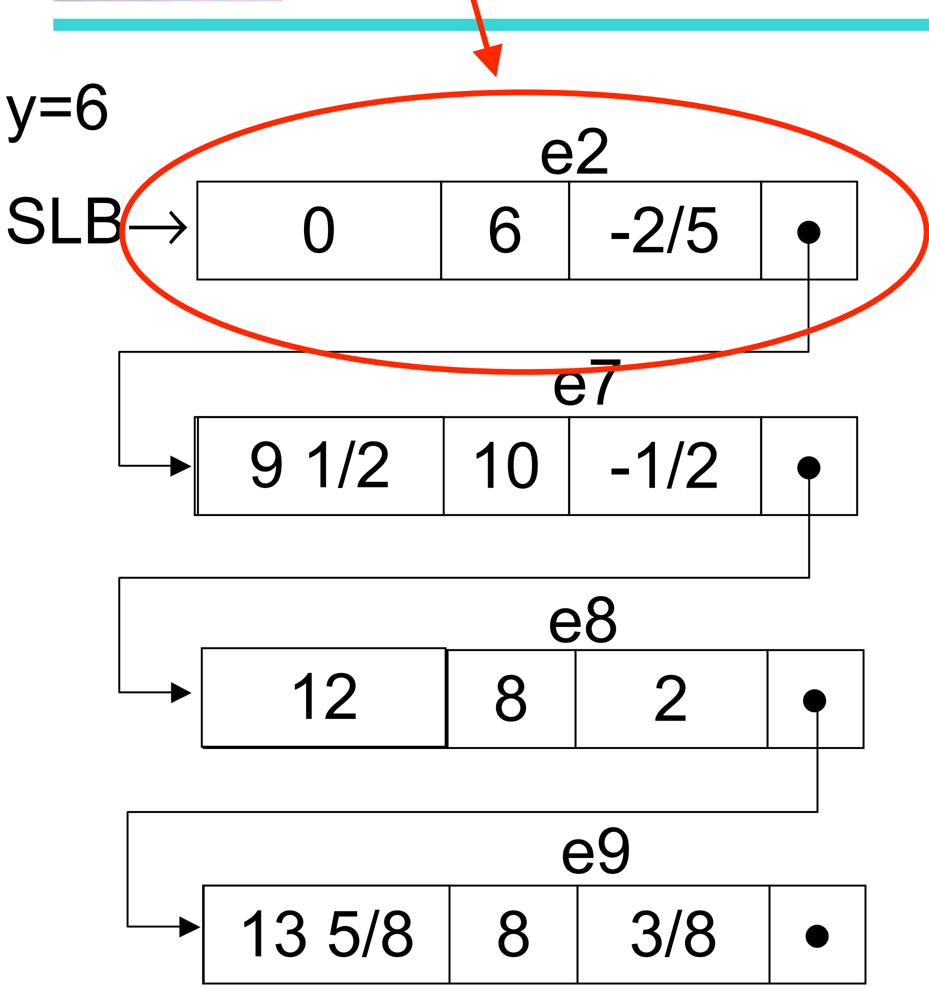


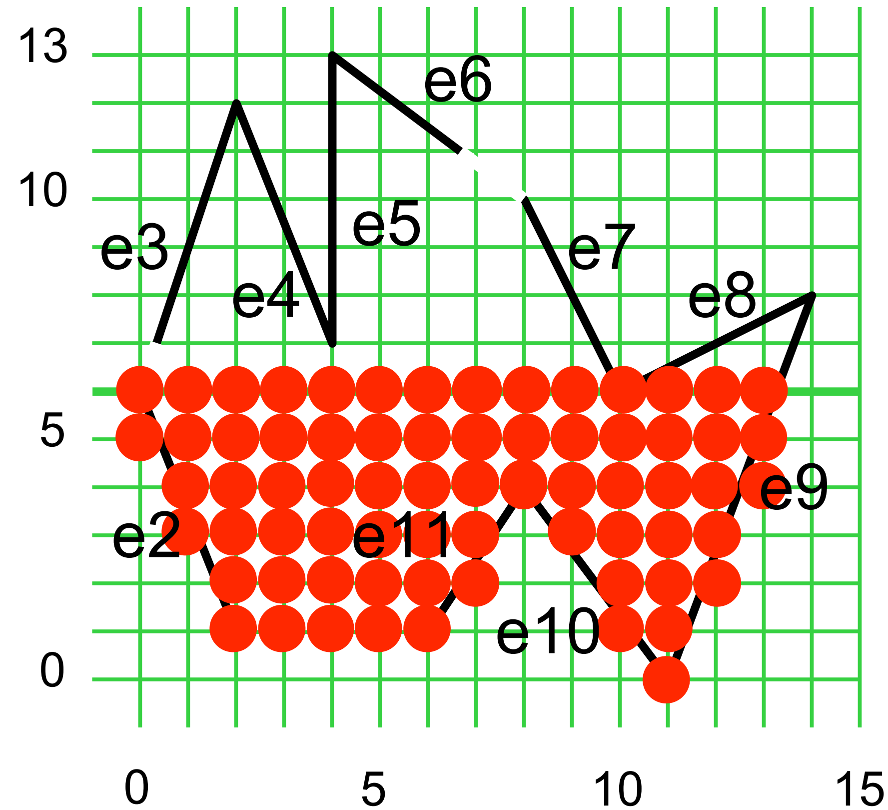


Running the Algorithm



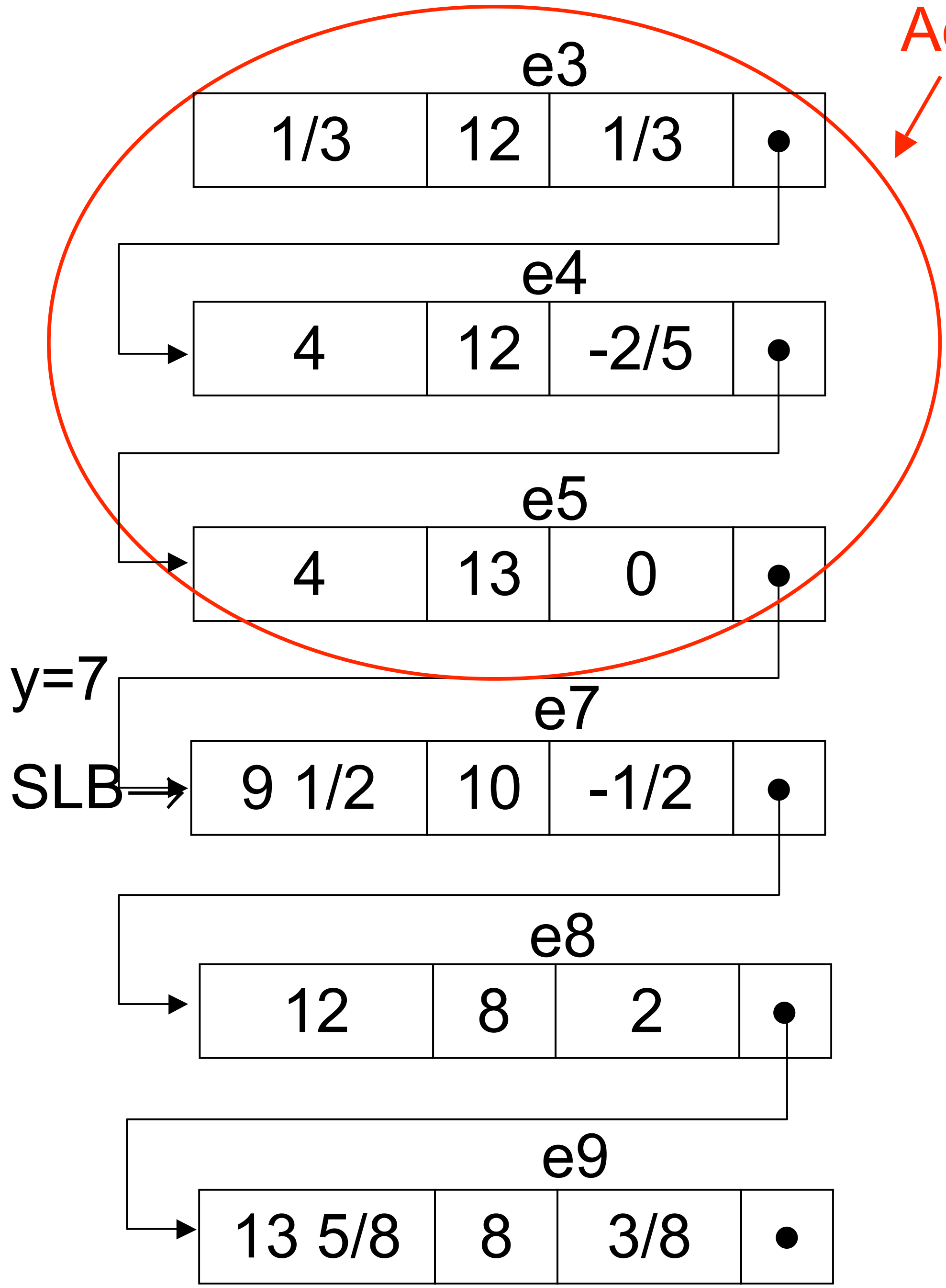
Remove this edge.

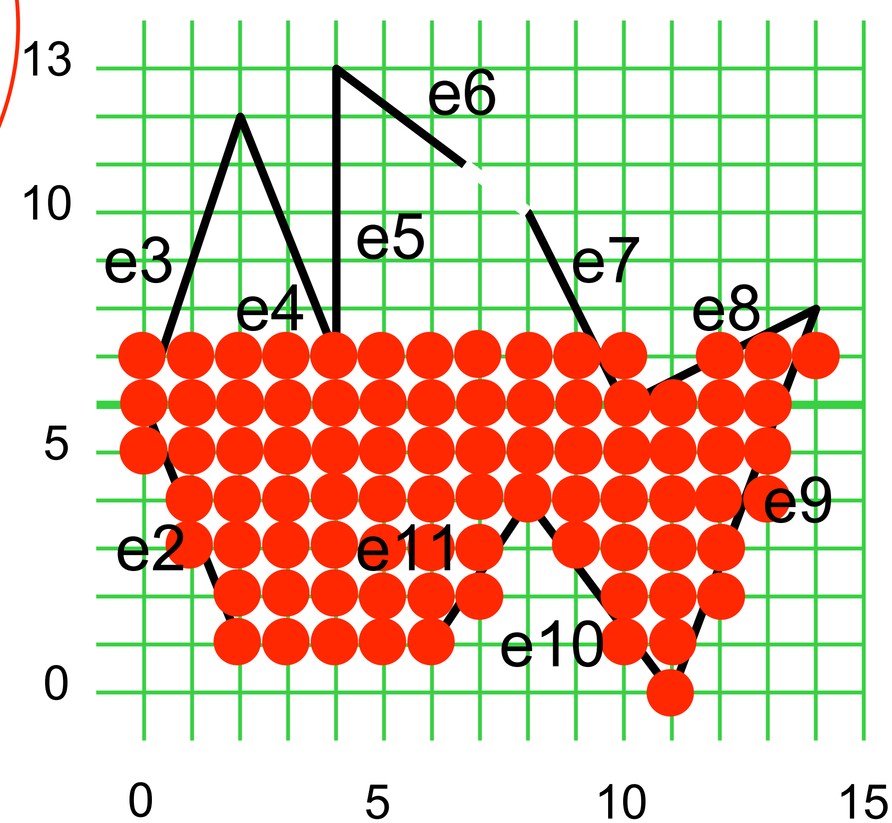




Running the Algorithm

Add these edges.





ET – the Edge Table

The **E**dge**T**able is for all nonhorizontal edges of all polygons.

ET has buckets based on edges smaller y-coordinate.

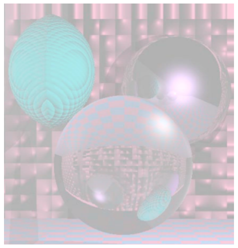
Edge Data:

- x-coordinate of smaller y-coordinate
- y-top
- 1/m = delta x
- polygon identification #: which polygons the edge belongs to

Polygon **T**able

Polygon **T**able

- A, B, C, D of the plane equation
- shading or color info (e.g. color and N)
- *in* (*out*) boolean
  - initialized to false (= *out*) at start of scanline
- z – at lowest y, x



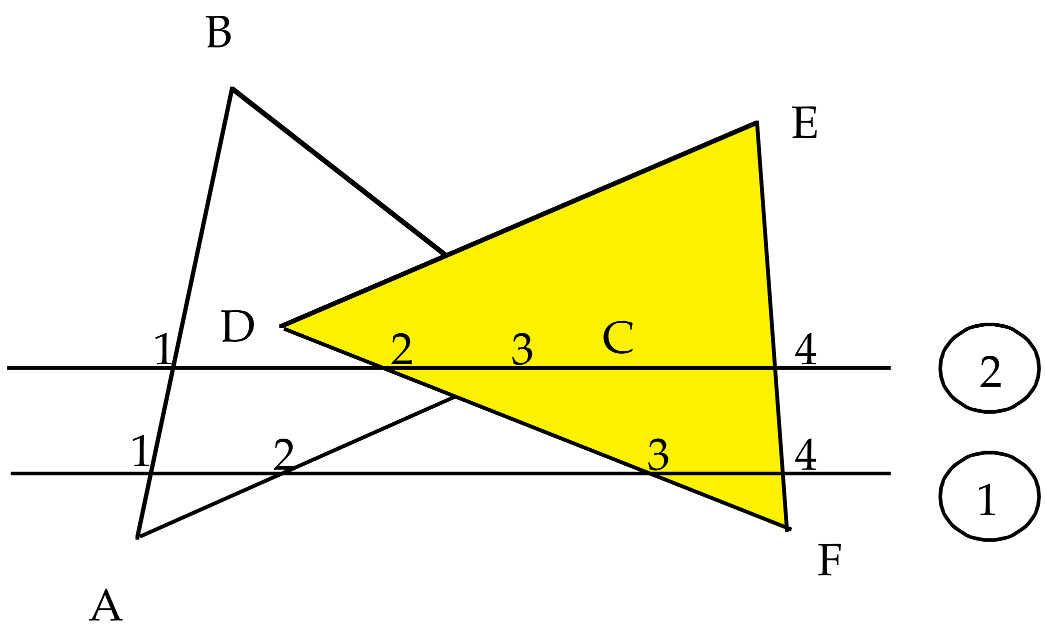
Coherence

- Non-penetrating polygons maintain their relative z values.
  - If the polygons penetrate, add a false edge.
- If there is no change in edges from one scanline to the next, and no change in order wrt x, then no new computations of z are needed.

Active Edge Table

Keep in order of increasing x.

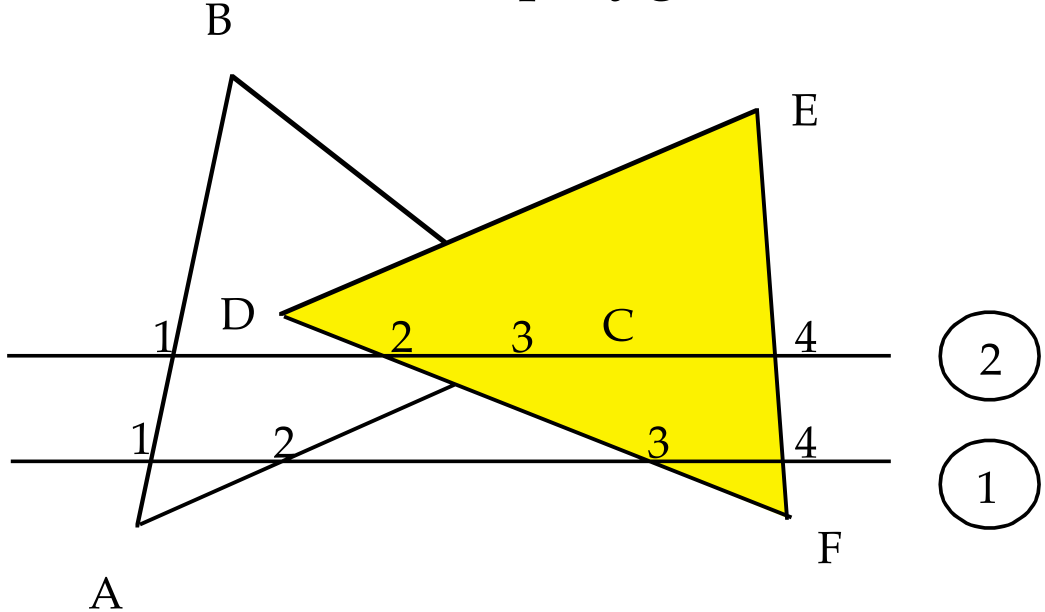
At (1) AET $\rightarrow$ AB $\rightarrow$ AC $\rightarrow$ DF $\rightarrow$ EF



Running the Algorithm 1

If more than one *in* is true, compute the z values at that point to see which polygon is furthest forward.

If only one *in* is true, use that polygon's color and shading.



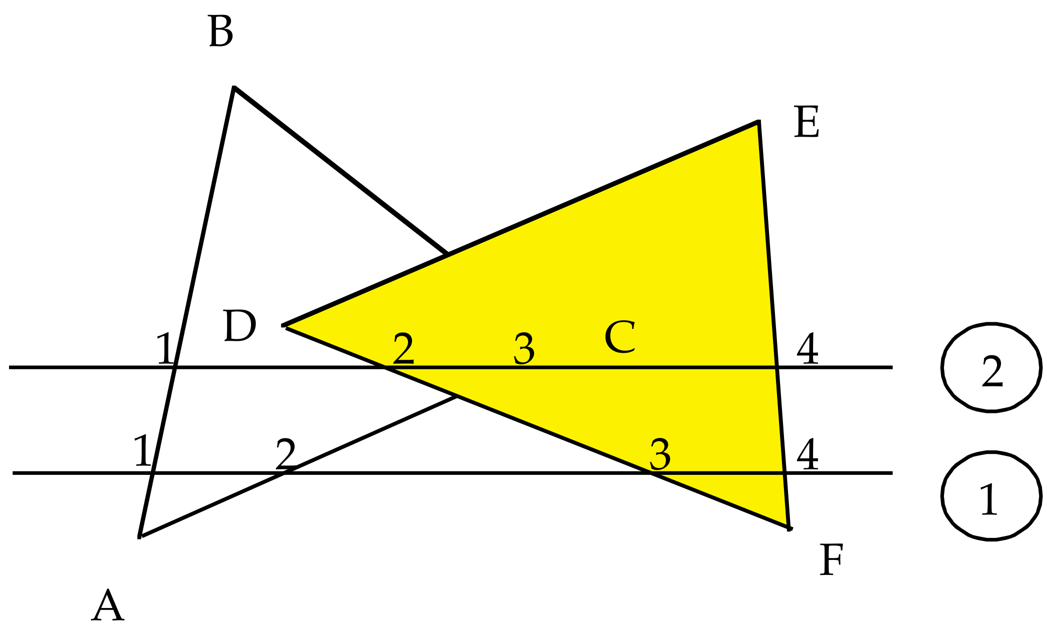
Running the Algorithm 2

On crossing an edge

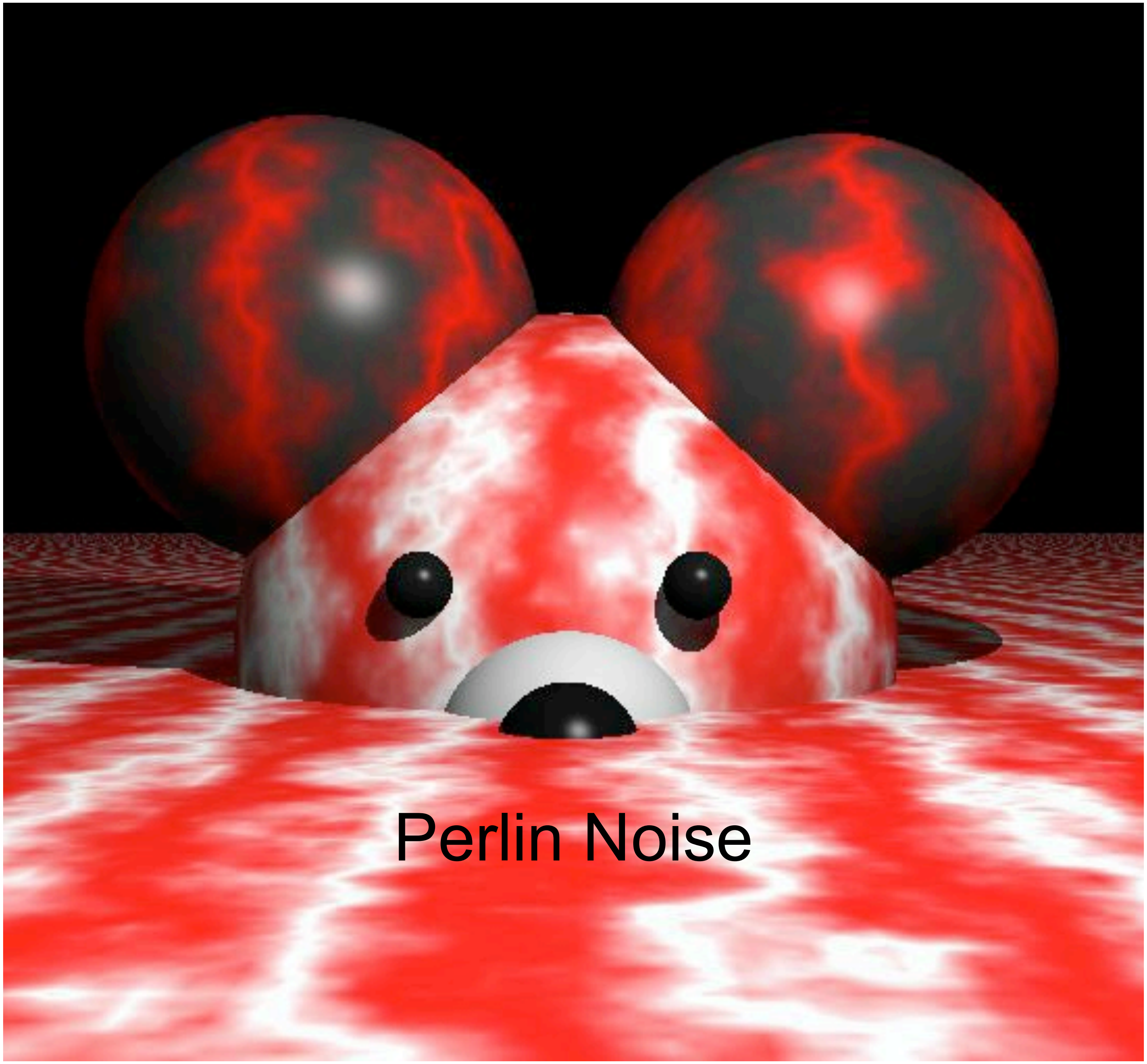
set *in* of polygons with that edge to **not** *in.*

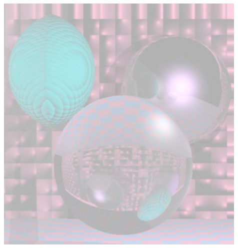
At (2) AET $\rightarrow$ AB $\rightarrow$ DF $\rightarrow$ AC $\rightarrow$ EF

If there is a third polygon, GHIJ behind the other two, after edge AC is passed at level (2) there is no need to evaluate z again - if the polygons do not pierce each other.



Time for a Break


Perlin Noise

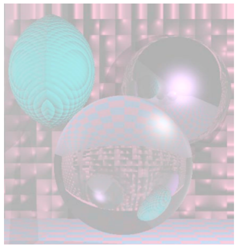


Noise Reference Links

- Perlin Noise by Ken Perlin
- Perlin Noise by Hugo Elias
- Perlin Noise and Turbulence by Paul Bourke

The Oscar™

To Ken Perlin for the development of Perlin Noise, a technique used to produce natural appearing textures on computer generated surfaces for motion picture visual effects.



The Movies

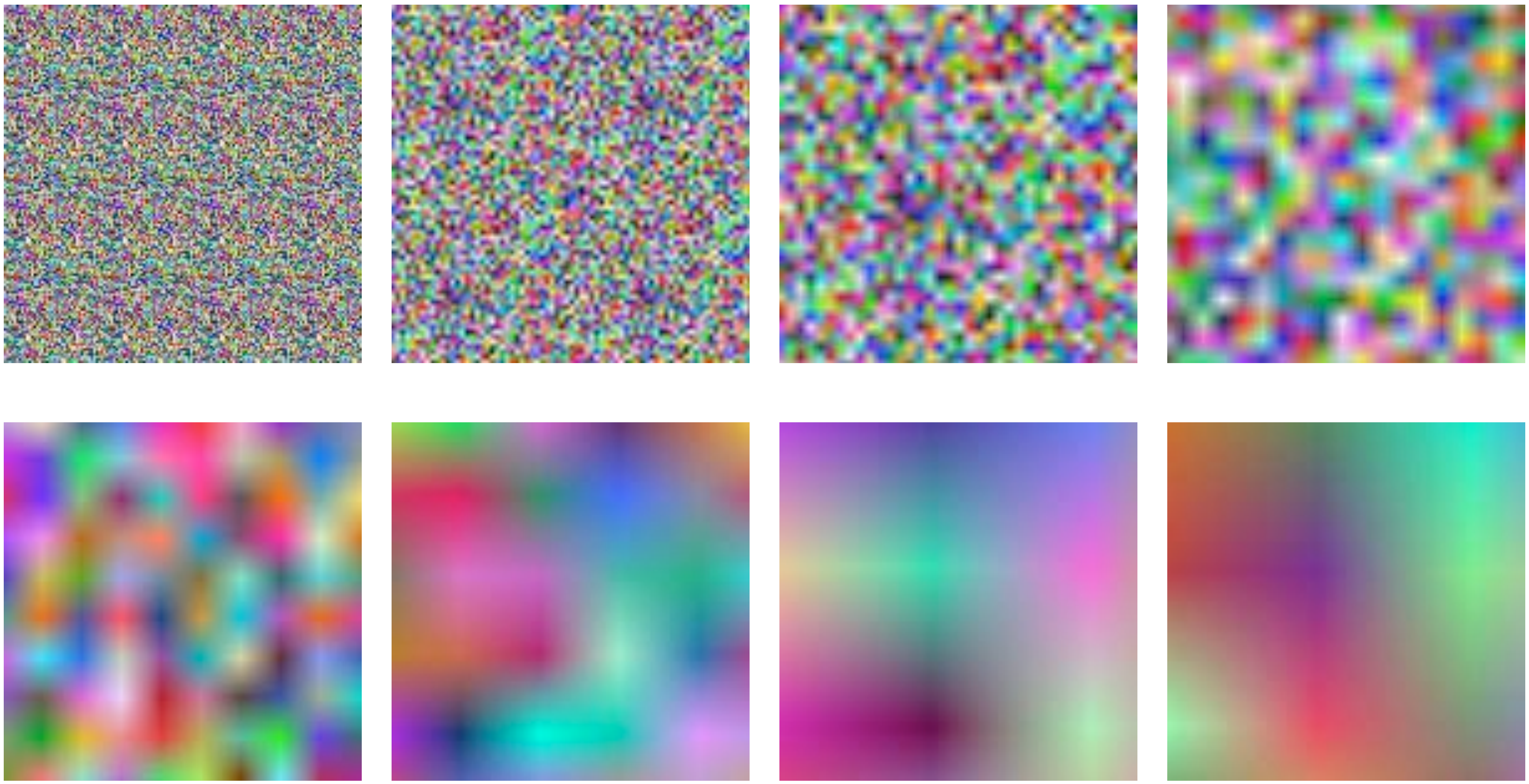
- James Cameron Movies (Abyss,Titanic,...)
- Animated Movies (Lion King, Moses,...)
- Arnold Movies (T2, True Lies, ...)
- Star Wars Episode I
- Star Trek Movies
- Batman Movies
- *and lots of others*

In fact, after around 1990 or so, *every* Hollywood effects film has used it.

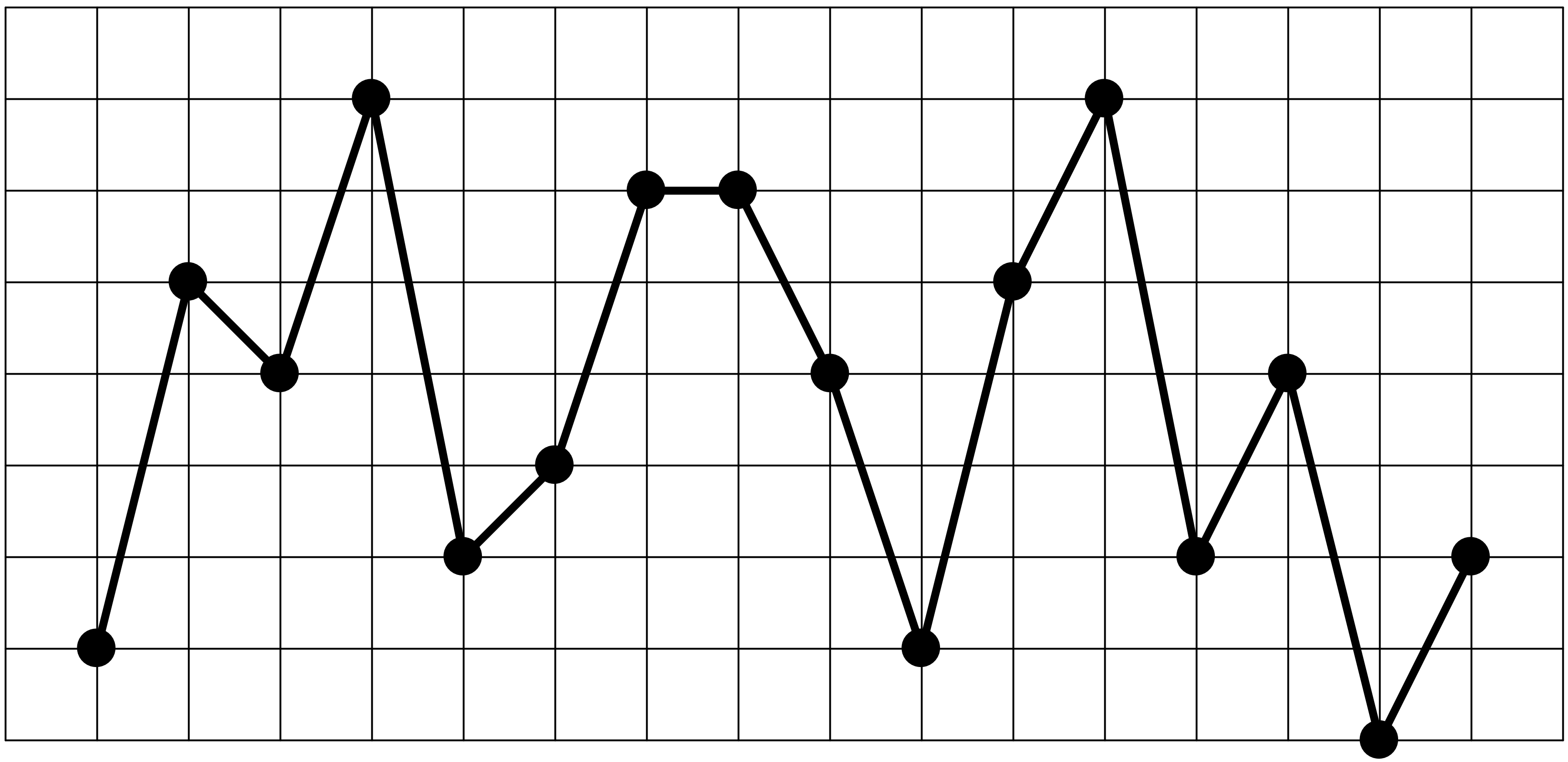
What is Noise?

- Noise is a mapping from R^n to R - you input an n-dimensional point with real coordinates, and it returns a real value.
- n=1 for animation
- n=2 cheap texture hacks
- n=3 less-cheap texture hacks
- n=4 time-varying solid textures

Noise is Smooth Randomness

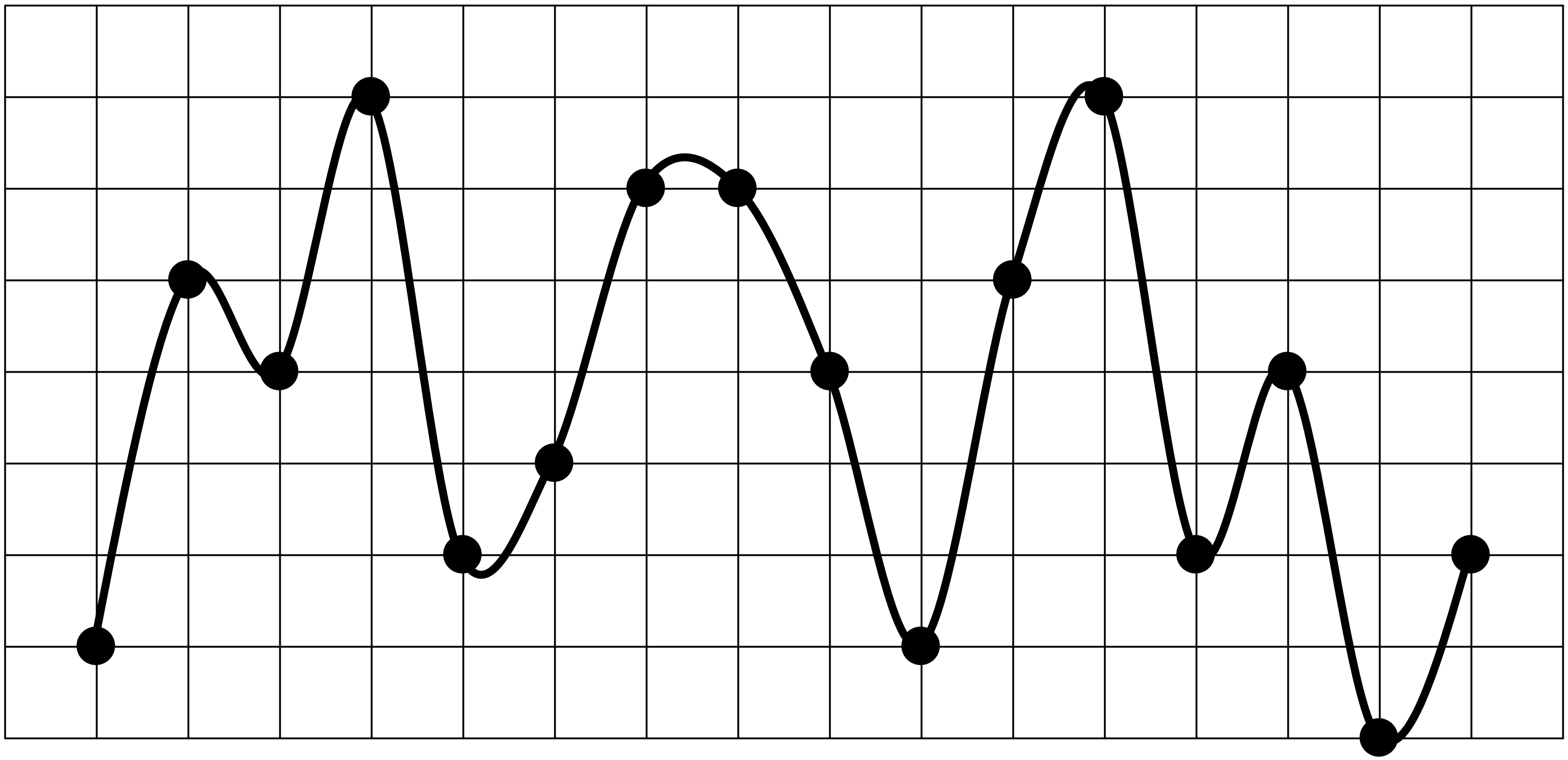


Making Linear Noise



1. Generate random values at grid points.
2. Interpolate linearly between these values.

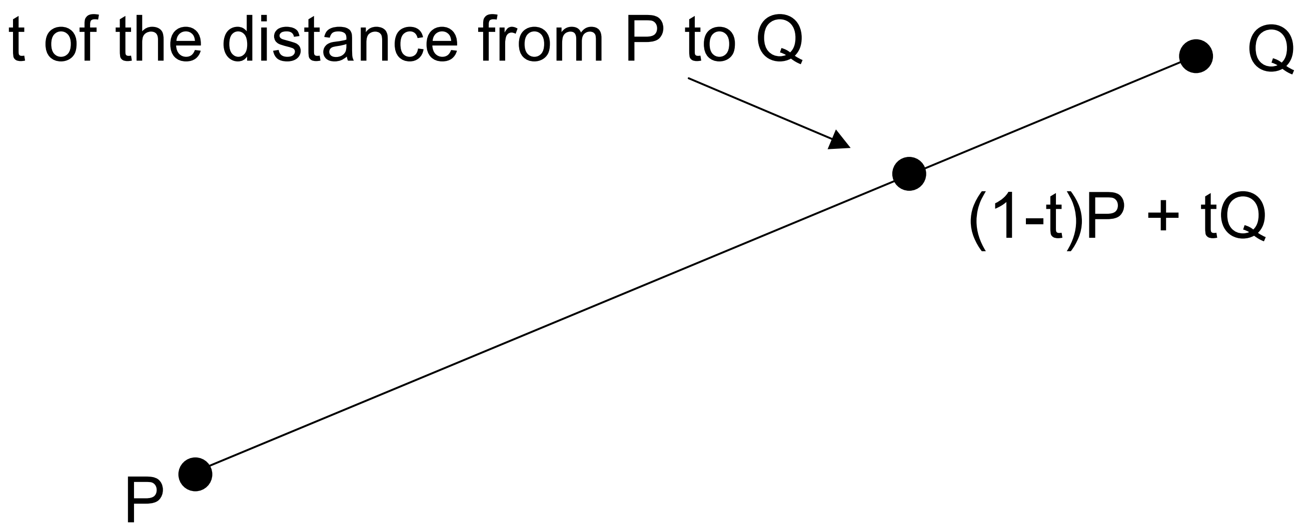
Making Splined Noise



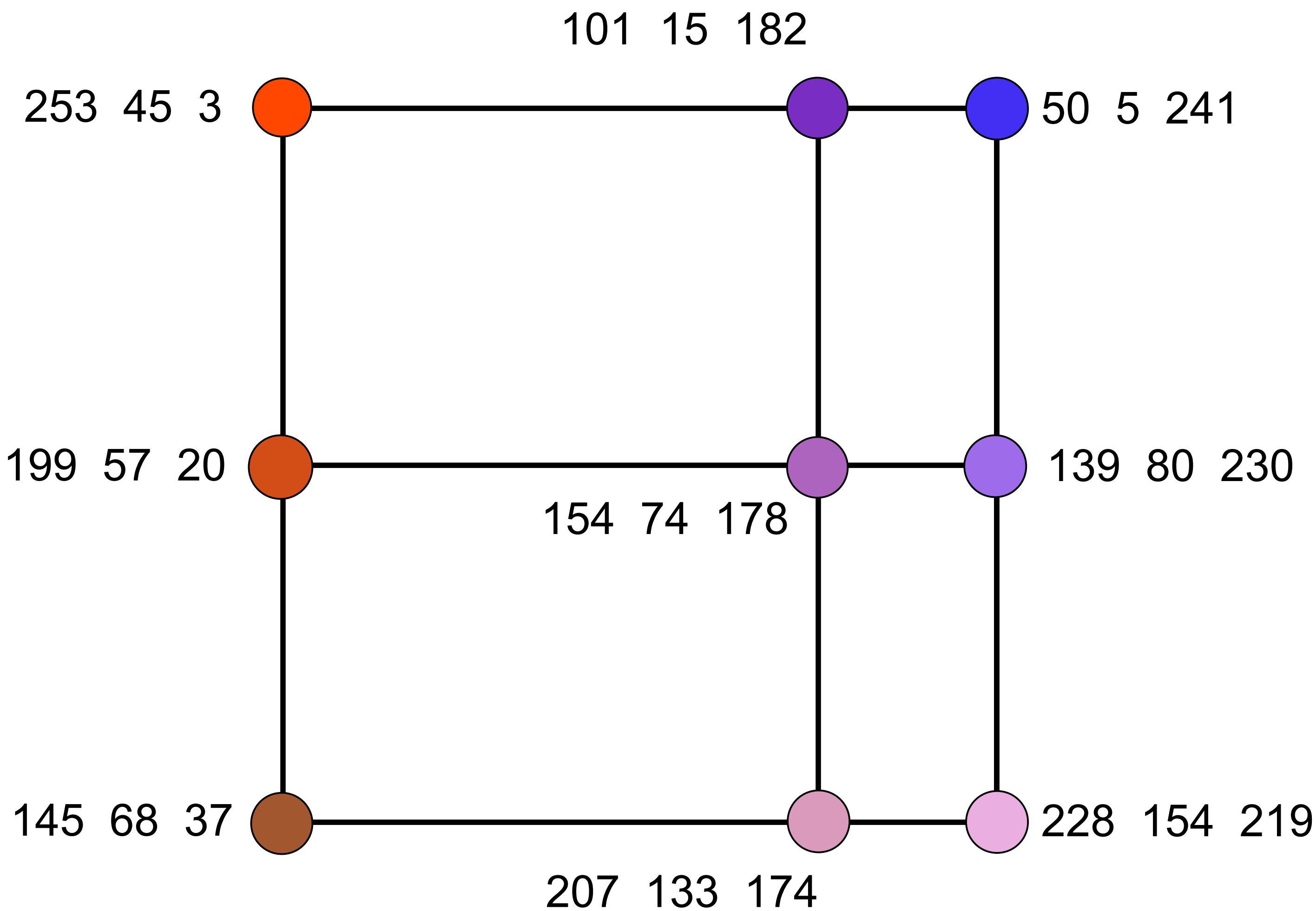
1. Generate random values at grid points.
2. Interpolate smoothly between these values.

lerping

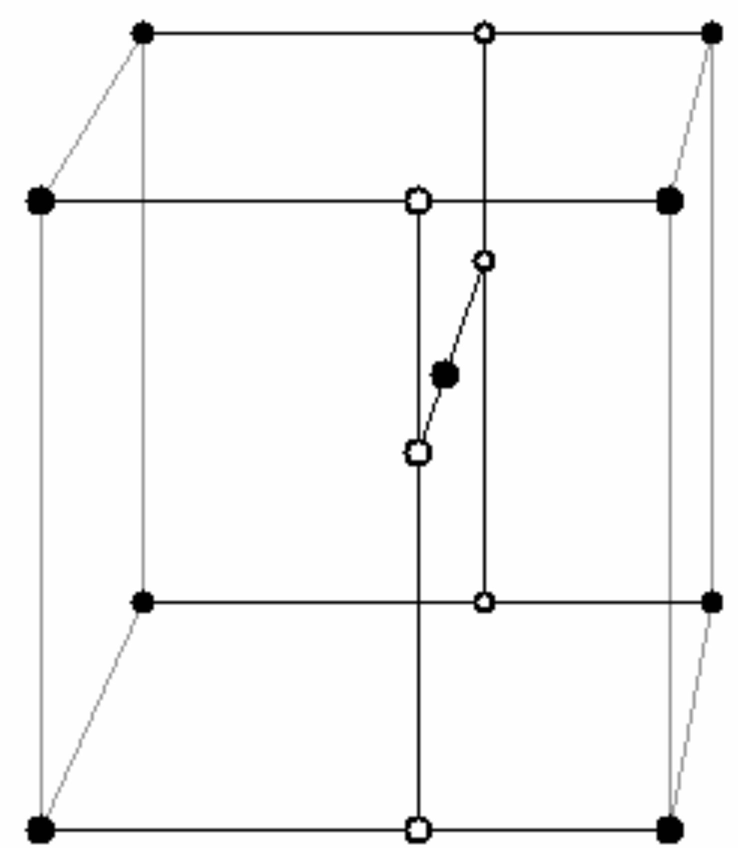
$$\text{lerp}(v1, v2, t) = (1 - t)v1 + tv2$$



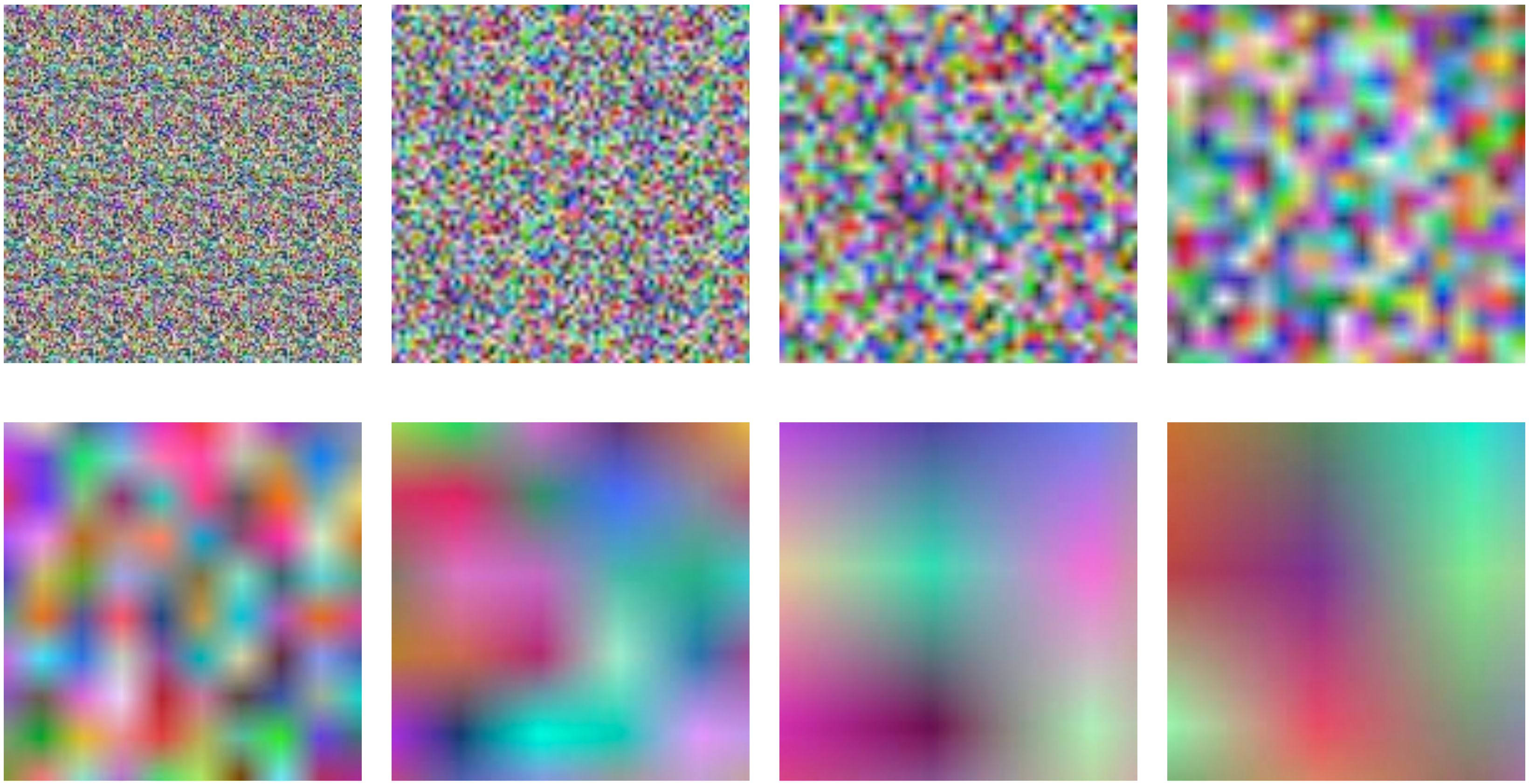
2D Linear Noise



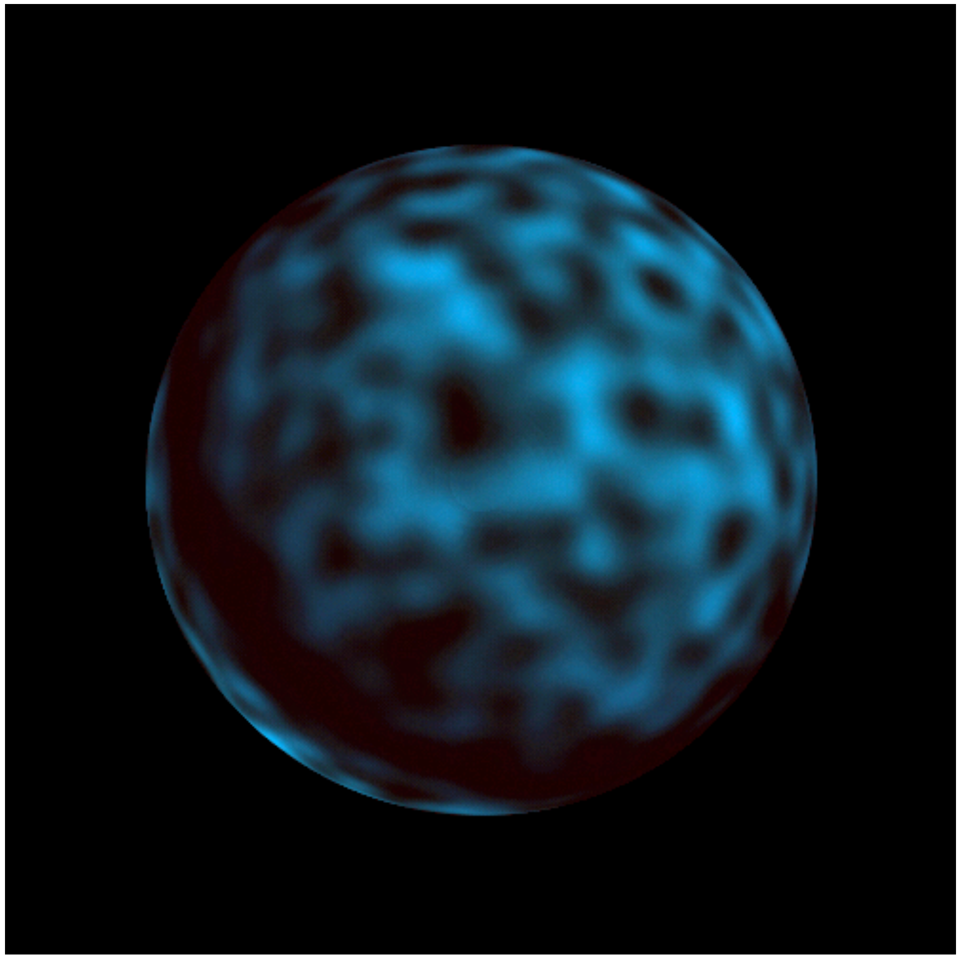
3D Linear Noise

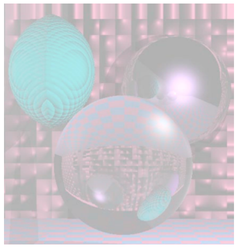


Noise is Smooth Randomness



Perlin Noise Sphere





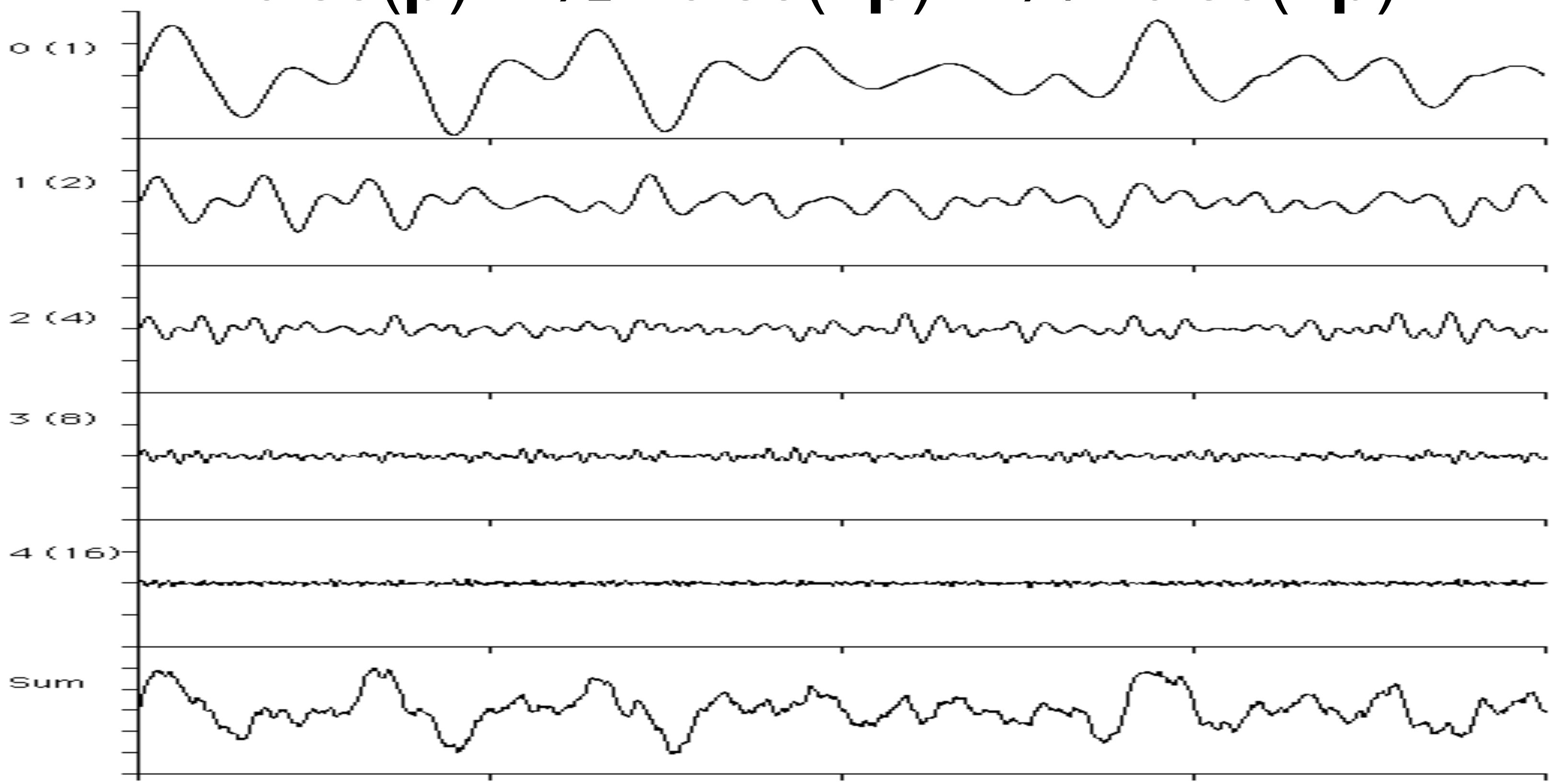
Noise Code

MATLAB Noise Code

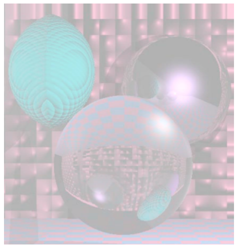
Don’t click this.

Turbulence or Sum $1/f^n$(noise)

noise(**p**) + ½ noise(2**p**) + ¼ noise(4**p**) ...



Perlin Noise and Turbulence by Baul Bourke



Turbulence and Persistence

$$\text{Turbulence}(x) = \sum_{i=0}^{n-1} p^i \,\mathrm{Noise}\left(b^i x\right)$$

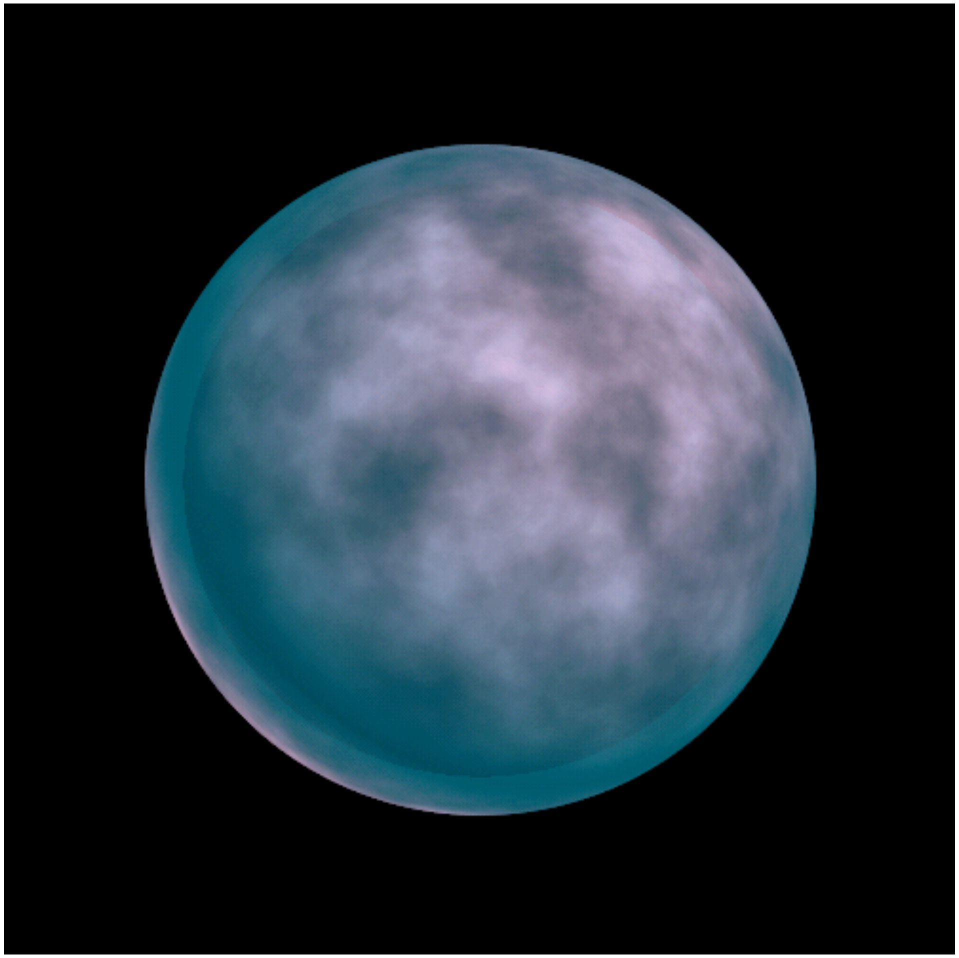
where n is the smallest integer such that $p^n <$ size of a pixel.

Usually $b = 2$.

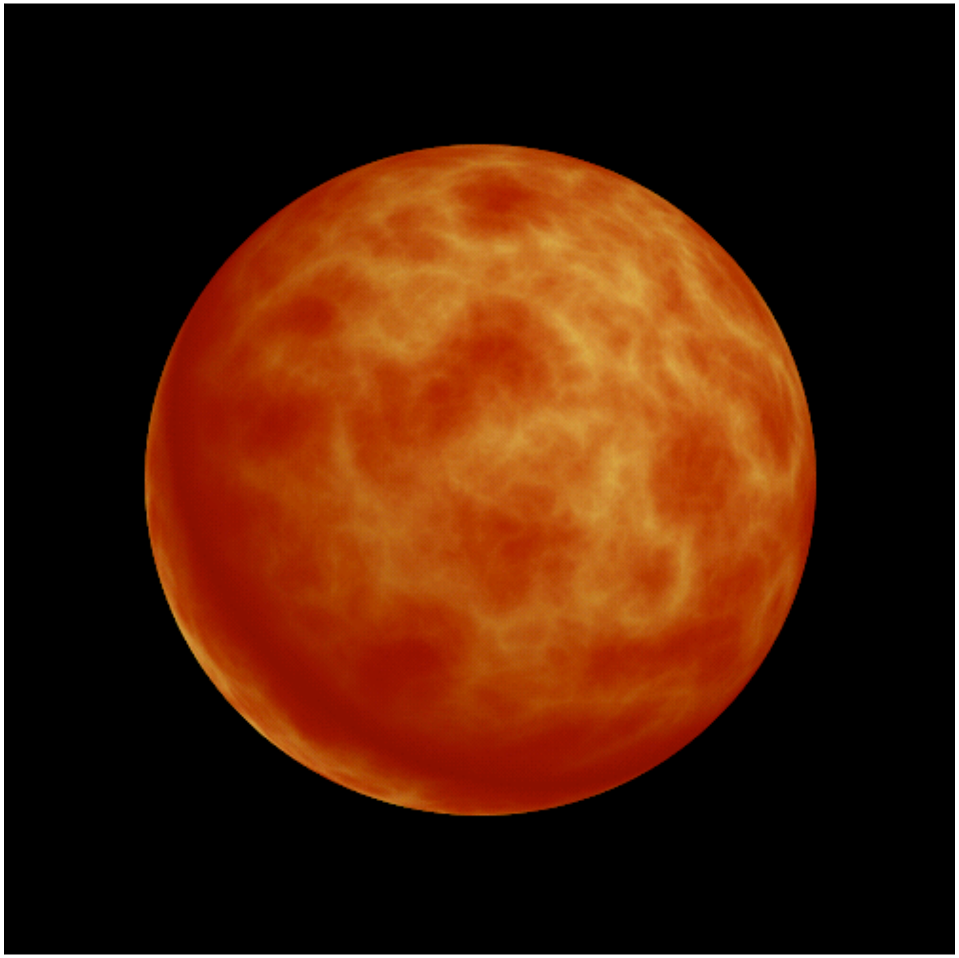
p is the *persistence,* $0 < p \leq 1$.

See Perlin Noise by Hugo Elias for more about persistence.

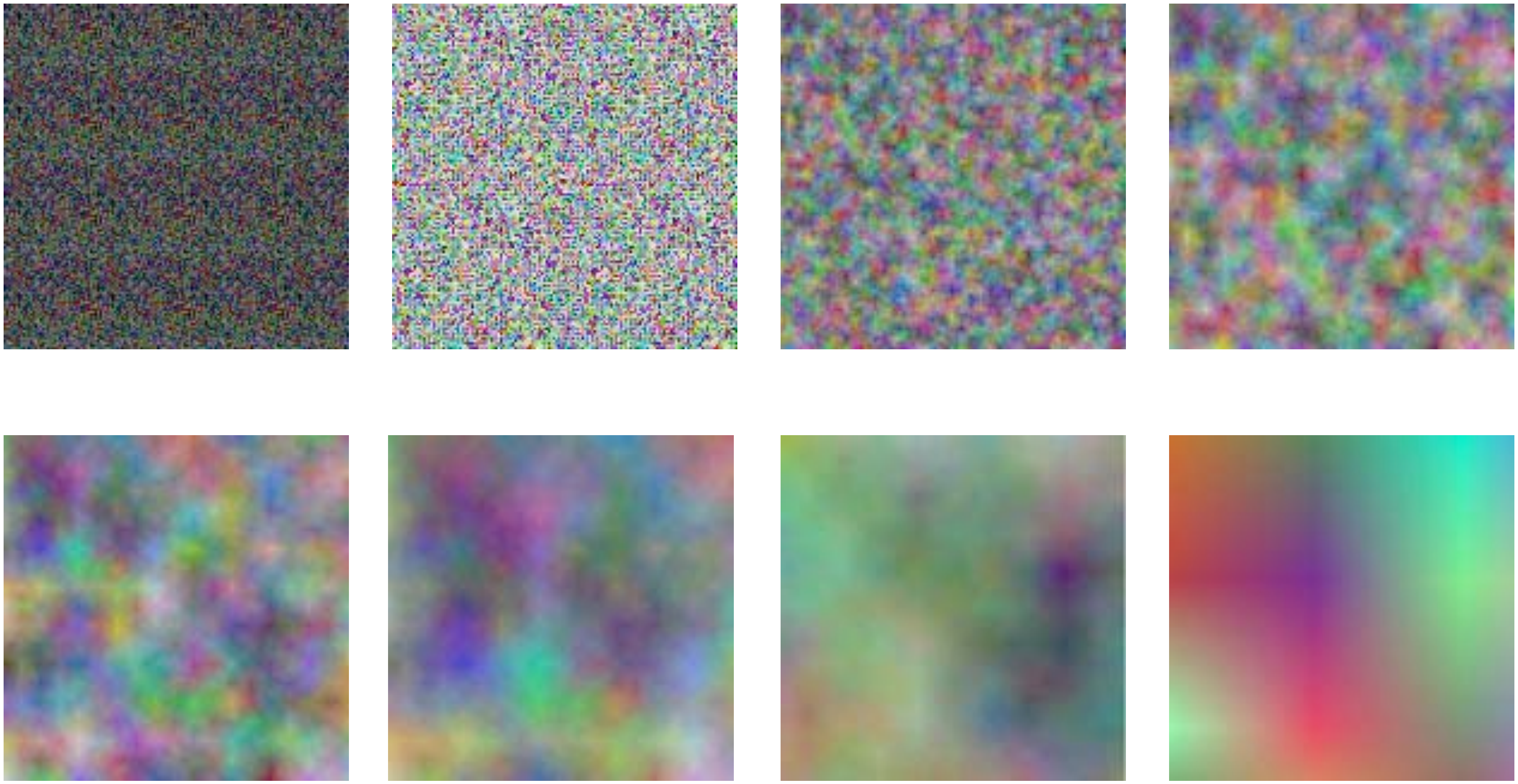
Perlin Sum 1/f(noise) Sphere



Perlin Sum 1/f(|noise|) Sphere

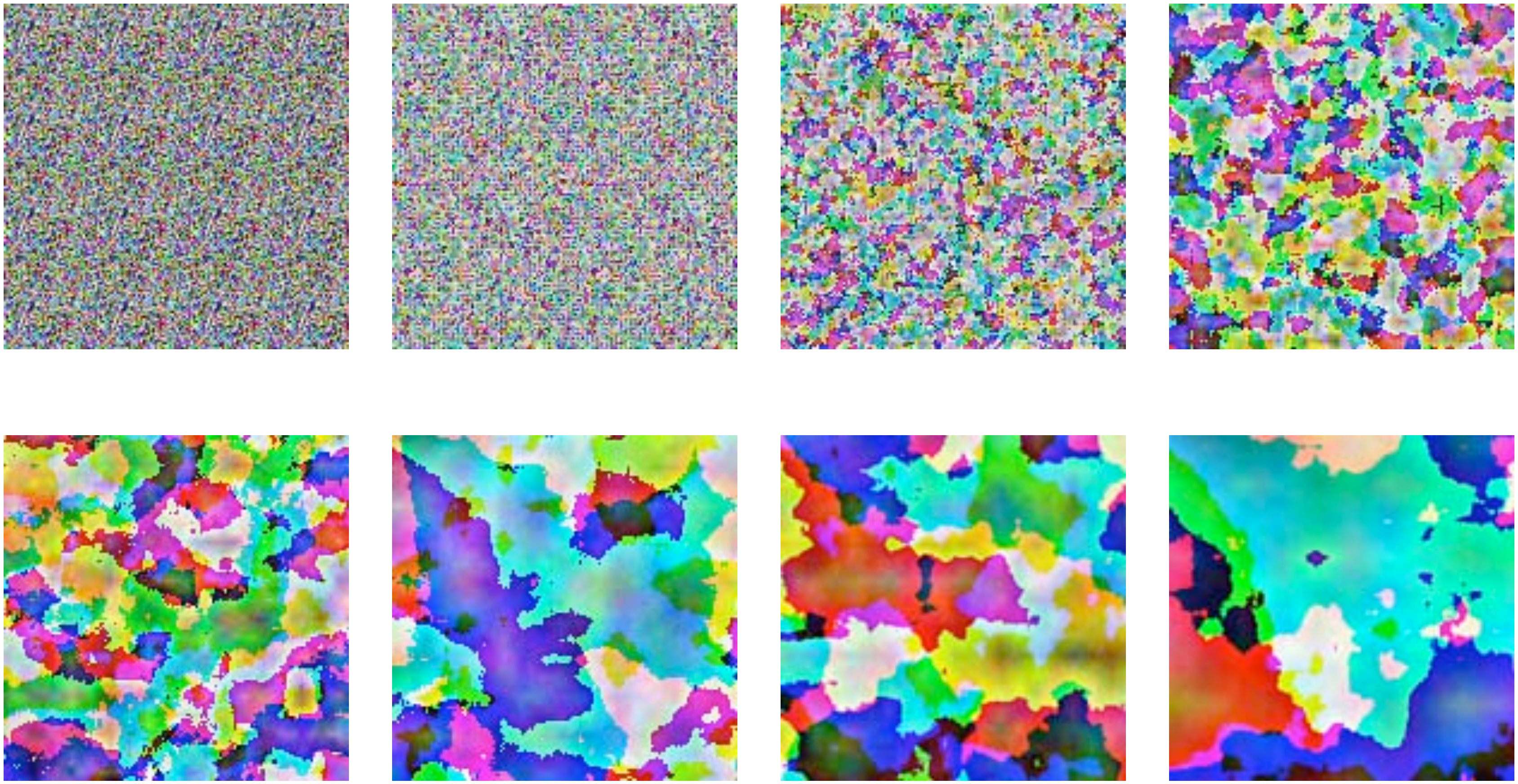


2D Nornalized Turbulence



Just Noise

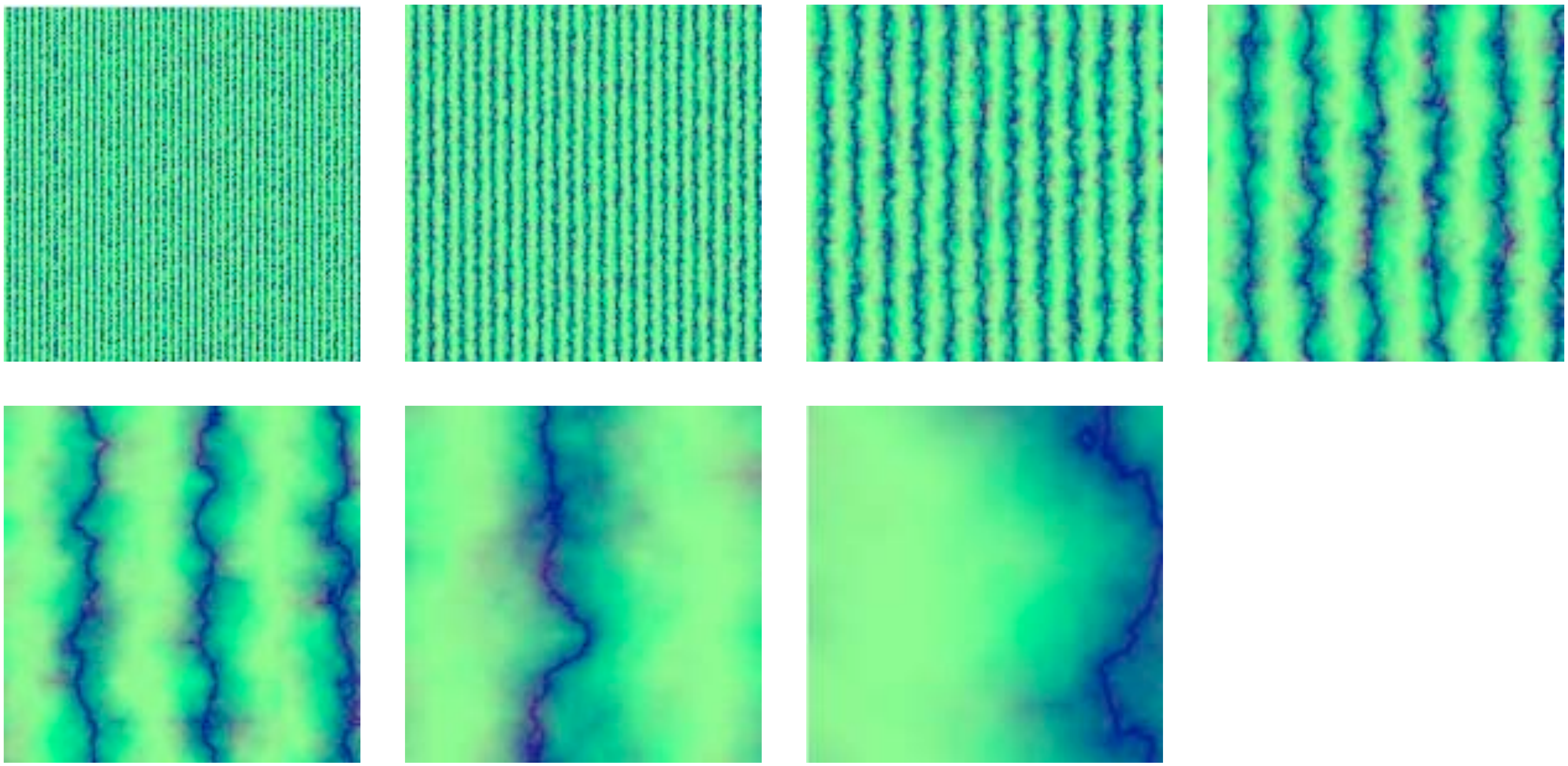
2D Turbulence



Turbulence Code

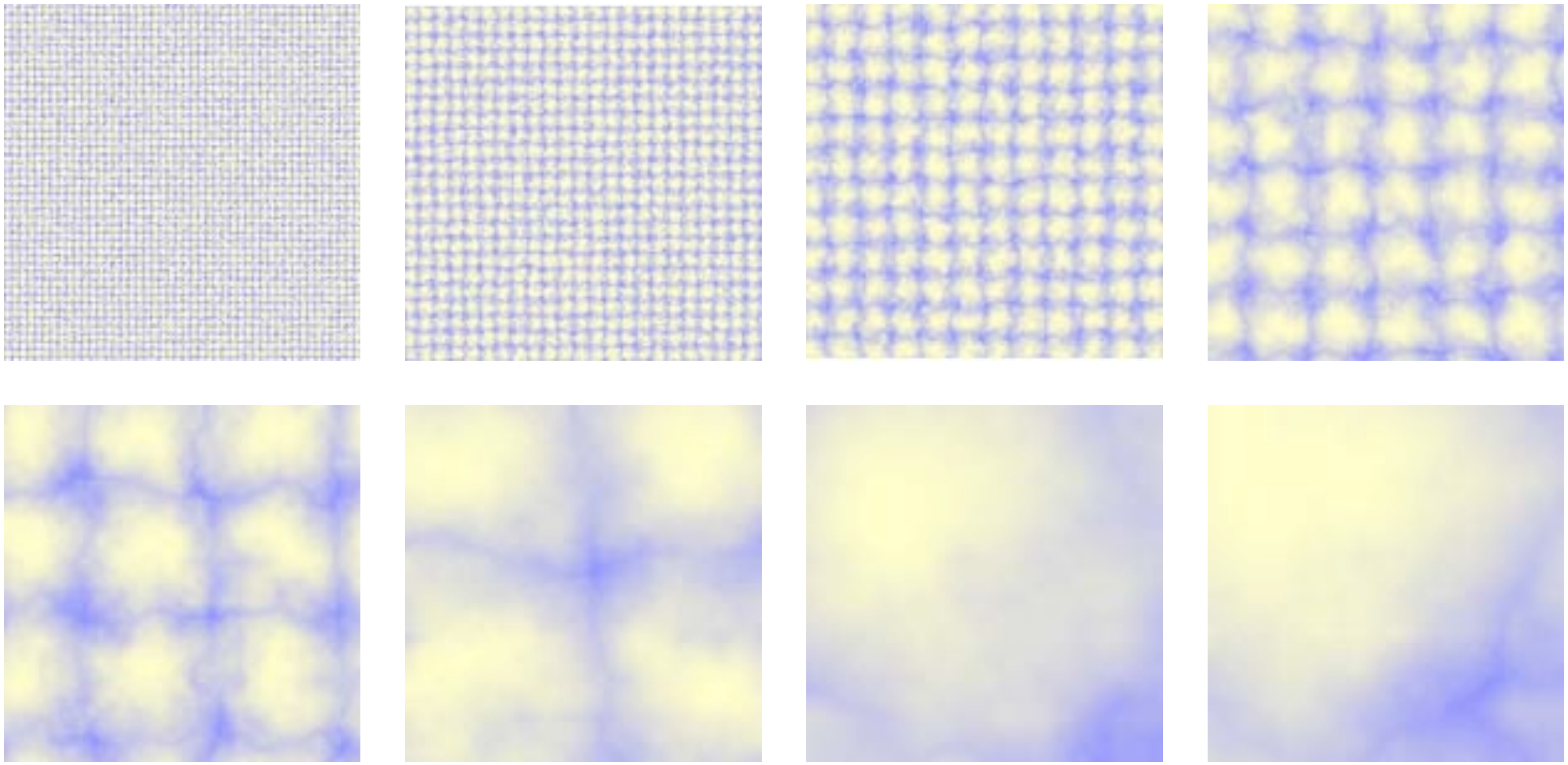
```
function turb = LinearTurbulence2(u, v, noise, divisor)
% double t, scale;
% LN(u, v) +LN(2u, 2v)/2 + LN(4u, 4v)/4 + ...
% Value is between between 0 and 2.

    t = 0;
    scale = 1;
    while (scale >= 1/divisor)
        t = t + linearNoise2(u/scale, v/scale, noise) * scale;
        scale = scale/2;
    end

    turb = t/2; % now value is between 0 and 1
```

Marble



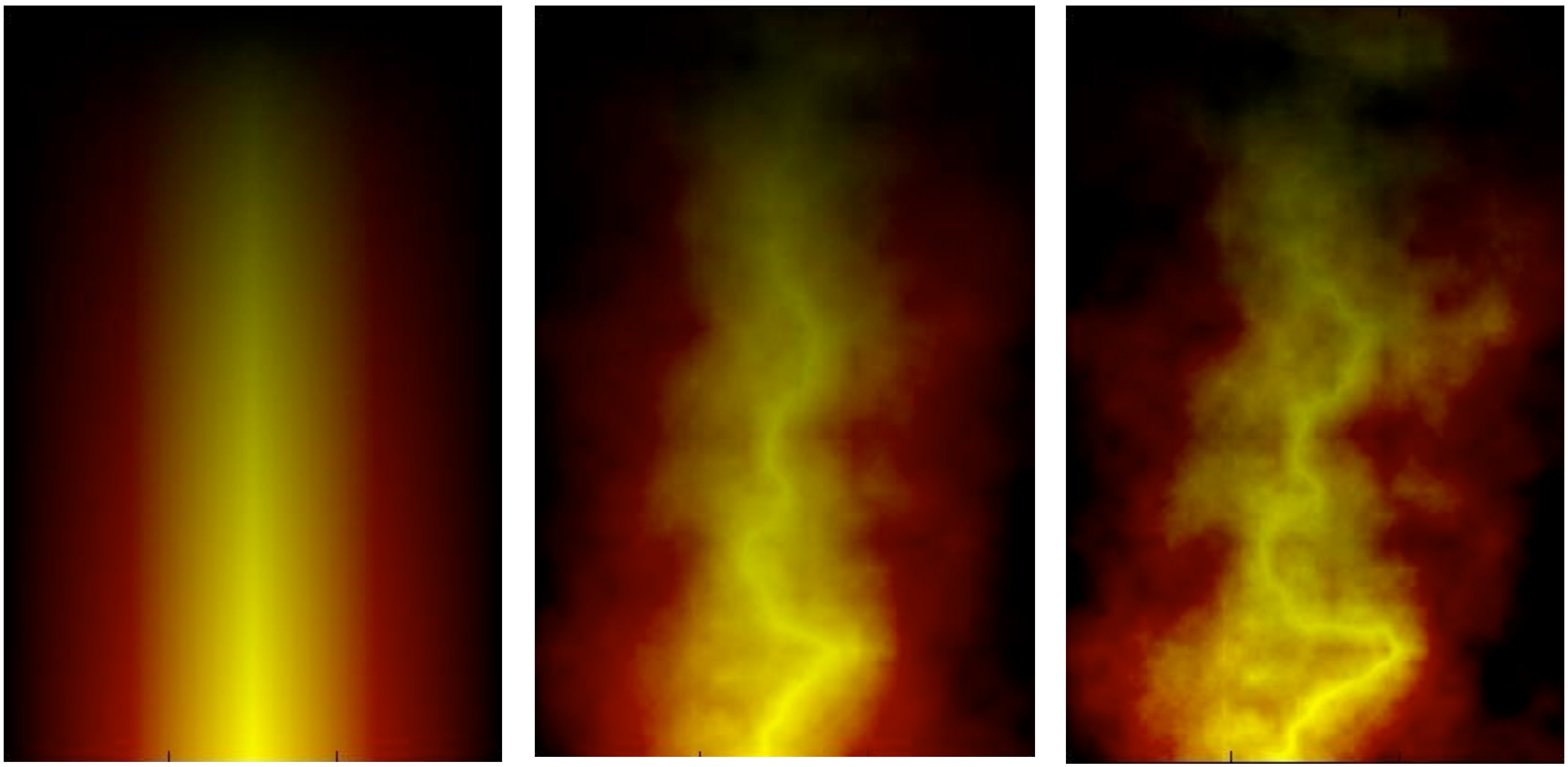
```
factorG  = sqrt(abs(sin(x + twist*turbulence(x, y, noise)
color = (0, trunc(factorG*255), 255);
```

Clouds



```
r = sqrt((x-200/d)*(x-200/d) + (y-200/d)*(y-200/d));
factorB = abs(cos(r + fluff*turbulence(x, y, noise));
color=(127 + 128*(1 - factorB), 127 + 128*(1 - factorB), 255);
```

Fire



Plane Flame Code

(MATLAB)

```
w = 300;     h = w + w/2;     x=1:w;     y=1:h;

flameColor = zeros(w,3); % Set a color for each x
flameColor(x,:)=…
          [1-2*abs(w/2-x)/w; max(0,1-4*abs(w/2-x)/w); zeros(1,w)]';

flame=zeros(h,w,3); % Set colors for whole flame
% 1 <= x=j <= 300=h, 1 <= y=451-i <= 450=h+h/2
for i = 1:h
   for j = 1:w
      flame(i,j,:)=(1-(h-i)/h)*flameColor(j,:);
   end
end
```

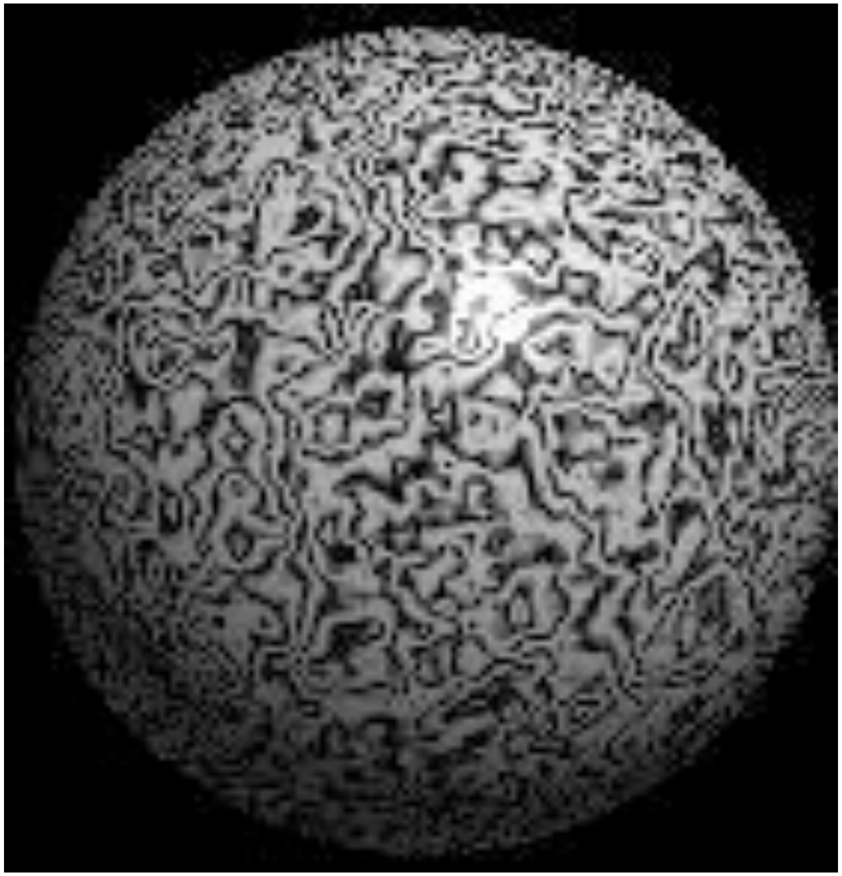
Turbulent Flame Code

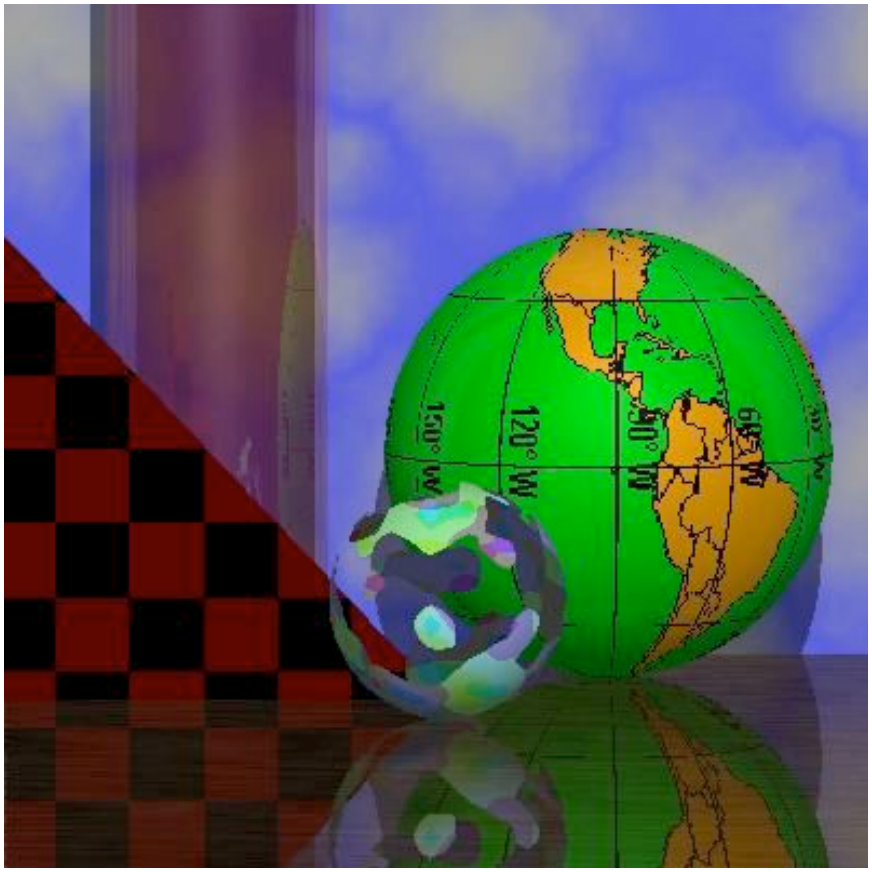
(MATLAB)

```
for u = 1:450
    for v = 1:300
        x = round(u+80*Tarray(u,v,1)); x = max(x,2); x = min(x,449);
        y = round(v+80*Tarray(u,v,2)); y = max(y,2); y = min(y,299);
        flame2(u,v,:) = flame(x,y,:);
    end
end
```

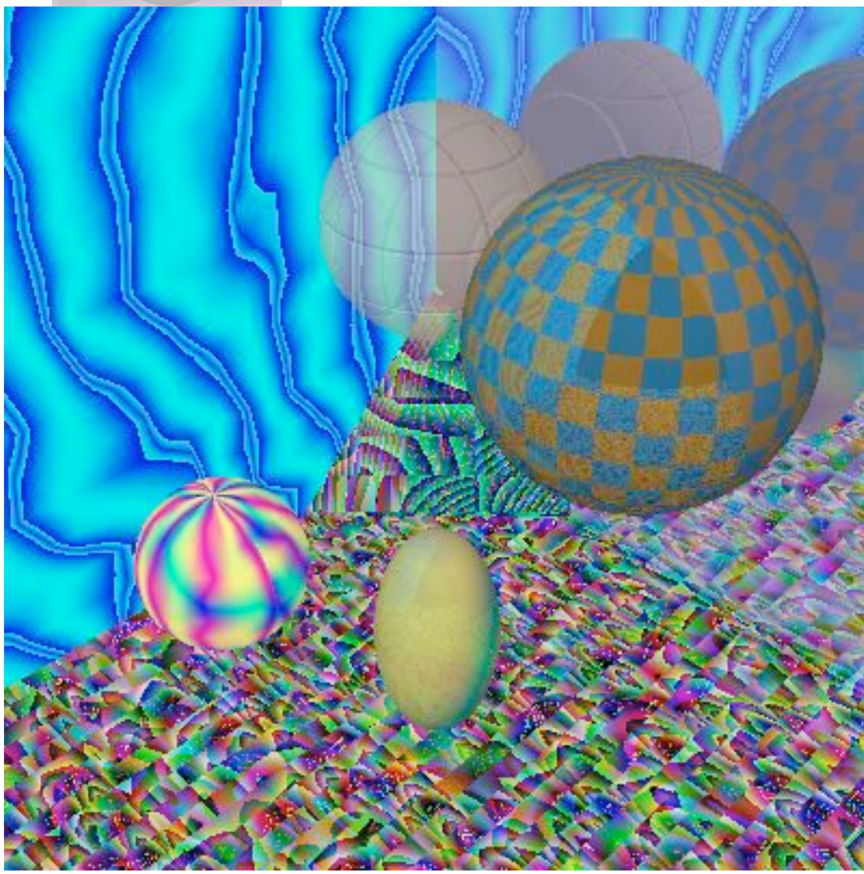
```
function Tarray = turbulenceArray(m,n)
noise1 = rand(39,39);
noise2 = rand(39,39);
noise3 = rand(39,39);
divisor = 64;
Tarray = zeros(m,n);

for i = 1:m
   for j = 1:n
      Tarray(i,j,1) = LinearTurbulence2(i/divisor, j/divisor, noise1, divisor);
      Tarray(i,j,2) = LinearTurbulence2(i/divisor, j/divisor, noise2, divisor);
      Tarray(i,j,3) = LinearTurbulence2(i/divisor, j/divisor, noise3, divisor);
   end
end
```

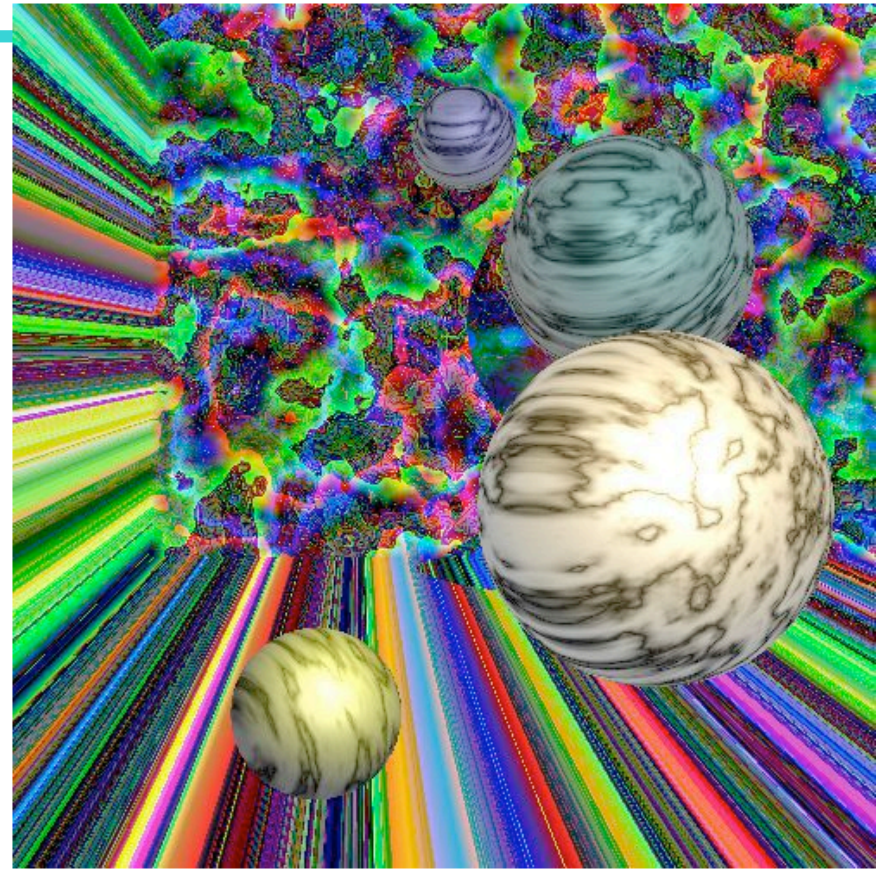
Student Images



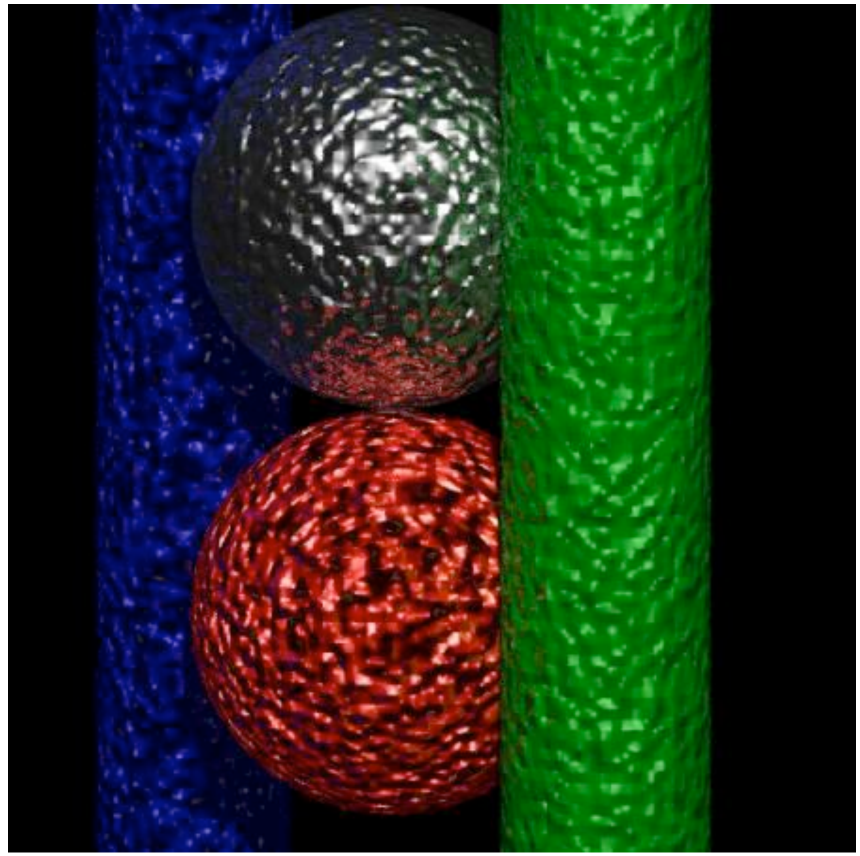


Student Images





Student Images



Perlin's Clouds and Corona