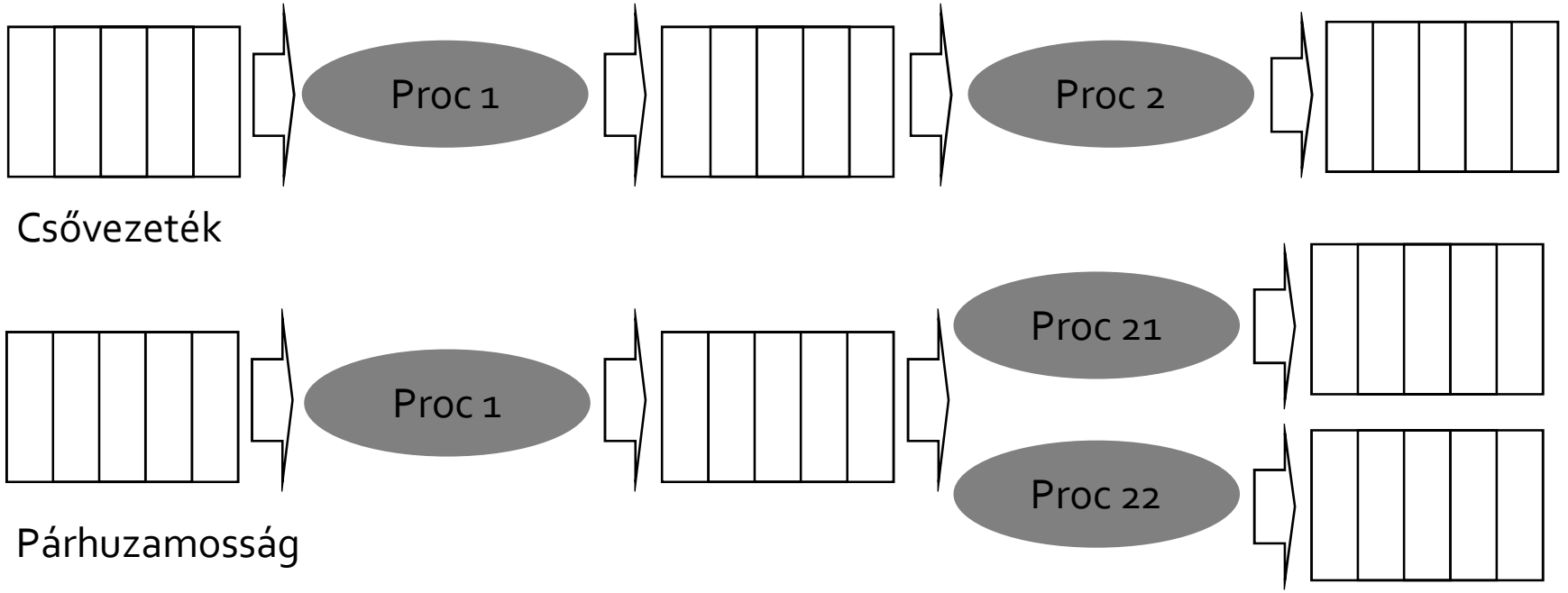
Adatfolyam és vektor feldolgozás

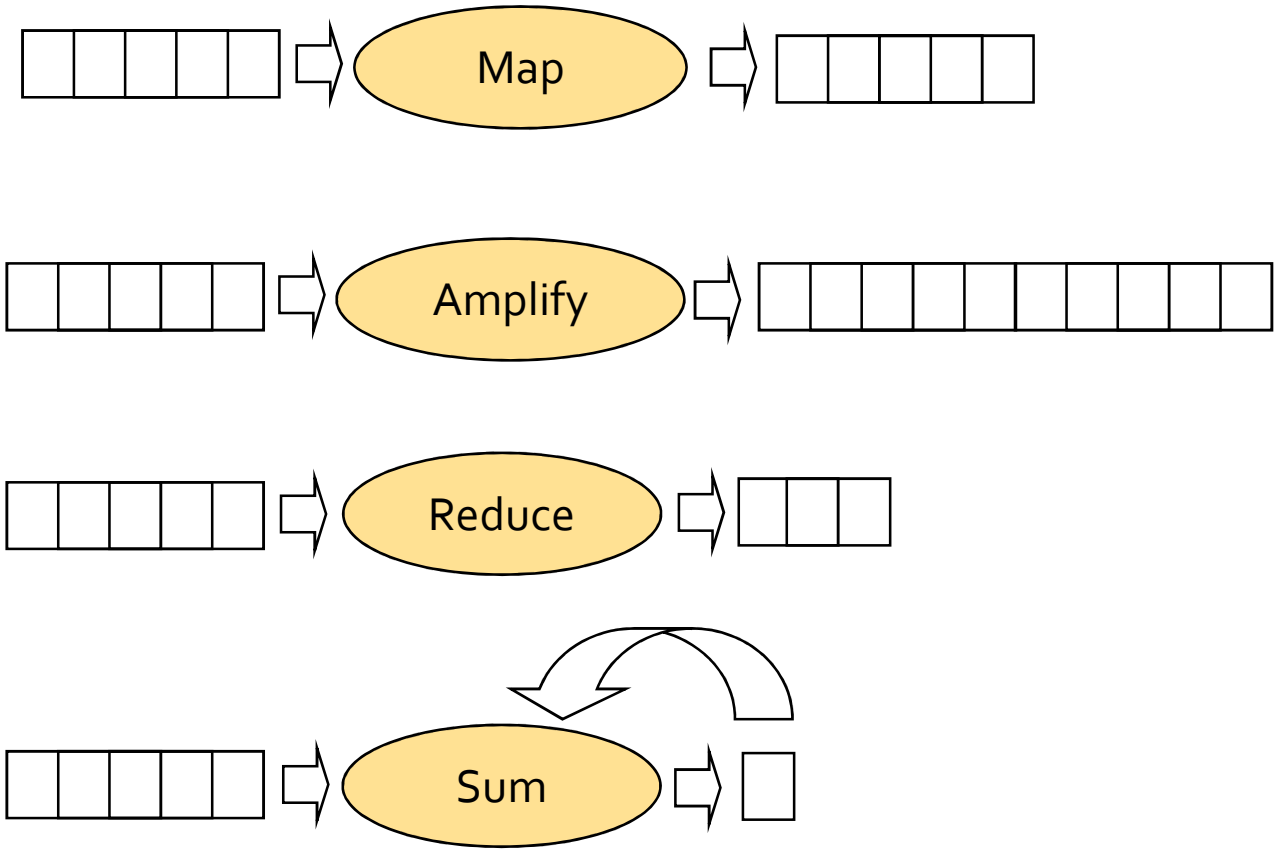
Adatfolyam feldolgozás

- Nincs szinkronizáció és kommunikáció
- Csővezeték alkalmazása
- Párhuzamosítás

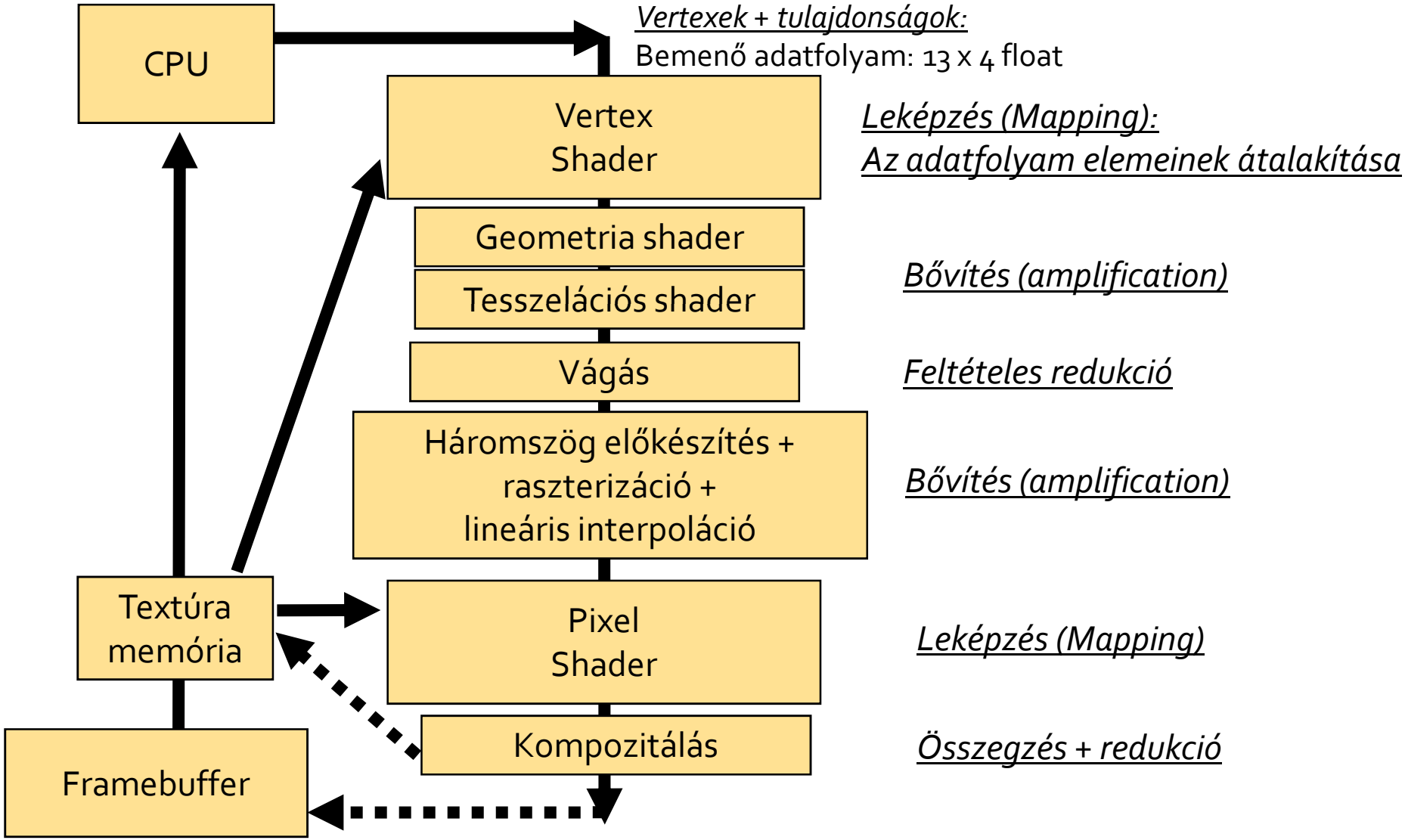


Adatfolyam feldolgozás

Alapműveletek



Adatfolyam feldolgozás



Párhuzamos feldolgozási sémák

- SISD (Single Instruction on Single Data)
  - Egymagos CPU, beágyazott vezérlők
- SIMD (Single Instruction on Multiple Data)
  - GPU Multiprocesszor
  - CPU kiterjesztések (MMX, SSE, 3DNow!)
- MIMD (Multiple Instruction on Multiple Data)
  - Több processzoros rendszerek
  - Több magos CPU
  - GPU

Vektor (Array) feldolgozás

- SIMD
  - Adatközpontú
  - Globális I/O igény
  - Erőteljesen párhuzamosítható
- Példa
  - két vektor összege
  - vektor mátrix szorzás

Vektor feldolgozás

- SIMD a CPU-n
  - Intel MMX
    - 8 speciális 64 bites regiszter
    - 8, 16, 32, 64 bites darabok
    - Egész műveletek
  - Intel SSE
    - 8-16 speciális 128 bites regiszter
    - Float, double, integer darabok
    - Egész és lebegő pontos műveletek

128 bits

xmm0
xmm1
xmm2
xmm3
xmm4
xmm5
xmm6
xmm7

Vektor feldolgozás

- SIMD a GPU-n
  - Bemenő adatfolyam
    - Vertex és tulajdonság streamek
    - Textúrák
  - Kimenő pixelhalmaz
  - Újra feldolgozható vertex és fragmens folyam

gozásra:

„Teljes képernyős” téglalap (CPU):

```
glViewport(0, 0, HRES, VRES)
glBegin(GL_QUADS);
glTexCoord2f(1,1); glVertex4f(-1,-1, 0, 1);
glTexCoord2f(1,0); glVertex4f(-1, 1, 0, 1);
glTexCoord2f(0,0); glVertex4f( 1, 1, 0, 1);
glTexCoord2f(0,1); glVertex4f( 1,-1, 0, 1);
glEnd( );
```

Vertex shader (GLSL):

```
in vec4 position;
in vec2 texCoord;
out vec2 fTexCoord;

void main (void) {
    gl_Position = position;
    fTexCoord = texCoord;
}
```

Fragment shader (GLSL):

```
in vec2 fTexCoord;
out vec4 outColor;

void main (void) {
    outColor = F(fTexCoord);
}
```

etria:
mszögek”

Az eredménytömb melyik elemeit számítjuk ki ?

Eredmény tömb

zintézis

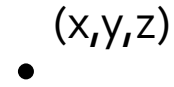
Kimeneti kép

neti tömbelemre
az algoritmus,
tokra: SIMD

Textúra vagy rasztertár

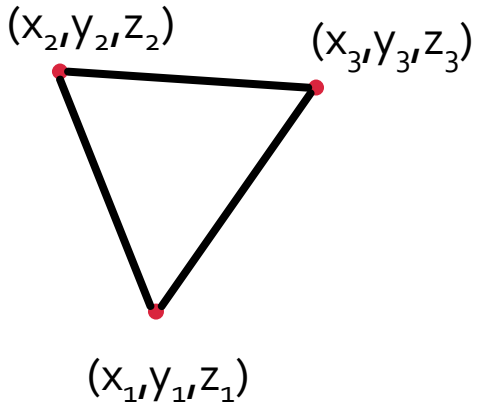
OpenGL: Geometria

- Primitívek
  - Pont (GL_POINTS)



  - Szakasz (GL_LINES)

  - Háromszög (GL_TRIANGLES)



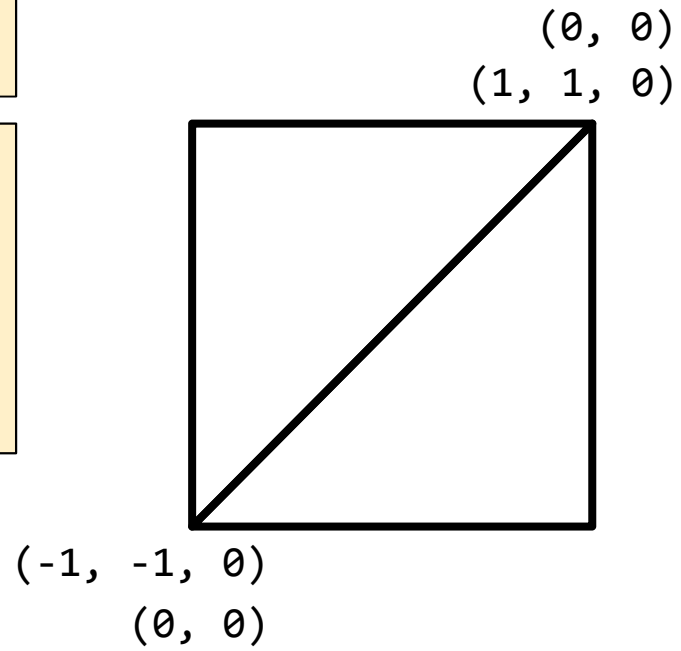
OpenGL: Geometria

- Vertex Array
  - Adat bufferek
    - 0.. GL_MAX_VERTEX_ATTRIBS
  - 4 komponensű vektor tömbök
  - Nincs meghatározott értelmezés

OpenGL: Geometria

```
GLfloat vertices[18] = { -1.0f,  1.0f, 0.0f,
                          1.0f,  1.0f, 0.0f,
                         -1.0f, -1.0f, 0.0f,
                         -1.0f, -1.0f, 0.0f,
                          1.0f,  1.0f, 0.0f,
                          1.0f, -1.0f, 0.0f };
```

```
GLfloat texCoords[12] = { 0.0f, 0.0f,
                          1.0f, 0.0f,
                          0.0f, 1.0f,
                          0.0f, 1.0f,
                          1.0f, 0.0f,
                          1.0f, 1.0f };
```



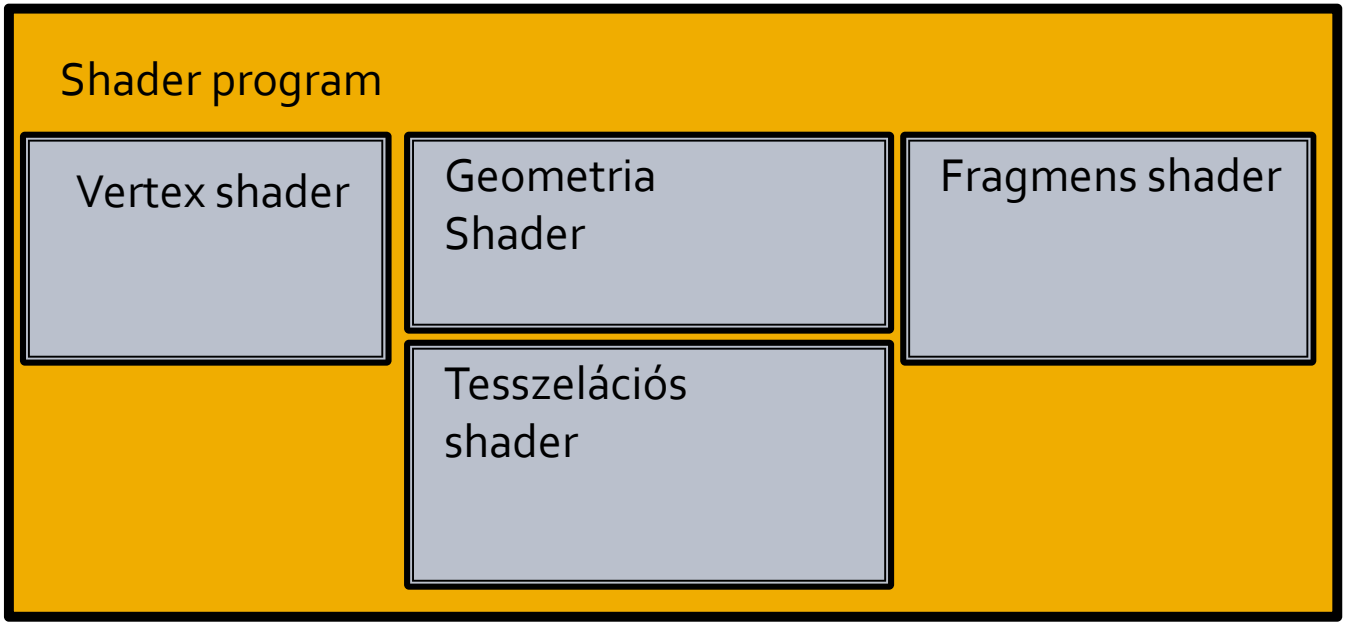
OpenGL: Geometria

```
GLuint vertexArray;
glGenVertexArrays(1, &vertexArray);
glBindVertexArray(vertexArray);

GLuint vertexBuffer;
glGenBuffers(1, &vertexBuffer);
glBindBuffer(GL_ARRAY_BUFFER, vertexBuffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(GLfloat) * vCount, vertices,
                                                        GL_STATIC_DRAW);
glVertexAttribPointer((GLuint)0, 3, GL_FLOAT, GL_FALSE, 0, 0);
...
glBindVertexArray(0);
```

```
glBindVertexArray(vertexArray);

shader->bindAttribLocation(0, „position”);
shader->bindAttribLocation(1, „texCoord”);

glDrawArrays(GL_TRIANGLES, 0, vCount);

glBindVertexArray(0);
```

OpenGL/GLSL

- Shader program
  - Egybe fogja a rendereléshez használt shadereket
  - Az OpenGL driver fordítja
  - Összeköti a shader változókat



OpenGL/GLSL

- Shaderek létrehozása

```
GLuint shader;
shader = glCreateShader(GL_VERTEX_SHADER);
glShaderSource(shader, 1, (const char**)&shaderSource, &length);
glCompileShader(shader);
GLuint errorFlag;
glGetShaderiv(shader, GL_COMPILE_STATUS, &errorFlag);
if(!errorFlag){
 glGetShaderInfoLog(...);
}
```

- Shader program

```
GLuint shaderProgram;
shaderProgram = glCreateProgram();
glAttachShader(shaderProgram, shader);
...
glLinkProgram(shaderProgram);
glGetProgramiv(shaderProgram, GL_LINK_STATUS, &errorFlag);
if(!errorFlag){
 glGetProgramInfoLog(...);
}
```

OpenGL/GLSL

- Program engedélyezése

```
glUseProgram(shaderProgram);
```

- Program tiltása

```
glUseProgram(0);
```

- Vertex attribútumok

```
glEnableVertexAttribArray(vertexArray);
glBindAttribLocation(shaderProgram, 0, „position”);
glBindAttribLocation(shaderProgram, 1, „texCoord”);
```

- Uniform paraméterek

```
GLuint location = glGetUniformLocation(shaderProgram, name);
glUniform1f(location, floatVal);
glUniform3f(location, floatValX, floatValY, floatValZ);
```

GLSL

- Adattípusok
  - Egyszerű típusok
    - bool, integer, float
  - Vektor típusok
    - vec2 texCoord
    - vec3 position
    - vec4 colorRGBA
    - bvec, ivec
  - Mátrix típusok
    - mat2, mat3, mat4
    - mat[2,3,4]x[2,3,4]

GLSL

- Minősítők
  - const: fordítási idejű konstans változó
    - const float maxIteration
  - uniform: globális változó a primitívre
    - uniform vec2 viewportSize
  - in: bemenő változó az előző shader lépcsőből
    - in vec2 texCoord
  - out: kimenő változó a következő lépcsőnek
    - out vec3 position

GLSL

- Operátorok
  - Aritmetika és bitműveletek
    - +,-,*,/,%, <<, >>, &, ^, |, ...
  - Adatkonverzió
    - (int)float, (float)bool, ...
  - Vektor és mátrix konstruktor
    - vec3(float), vec4(float, vec3), mat2(float)
  - Swizzle operátor
    - .{xyzw}, .{rgba}, .{stpq}
    - vec2 v2 = vec3(1.0, 1.0, 1.0).xy

GLSL

- Beépített függvények
  - Trigonometria és szögfüggvények
    - radians, degrees, sin, cos, atan, …
  - Geometriai függvények
    - length, distance, dot, cross, normalize, …
  - Exponenciális függvények
    - pow, exp, log, sqrt, …
  - Általános függvények
    - abs, sign, floor, ceil, mix, smoothstep, min, max, …
  - Mátrix függvények
    - transpose, determinant, inverse, …

GLSL

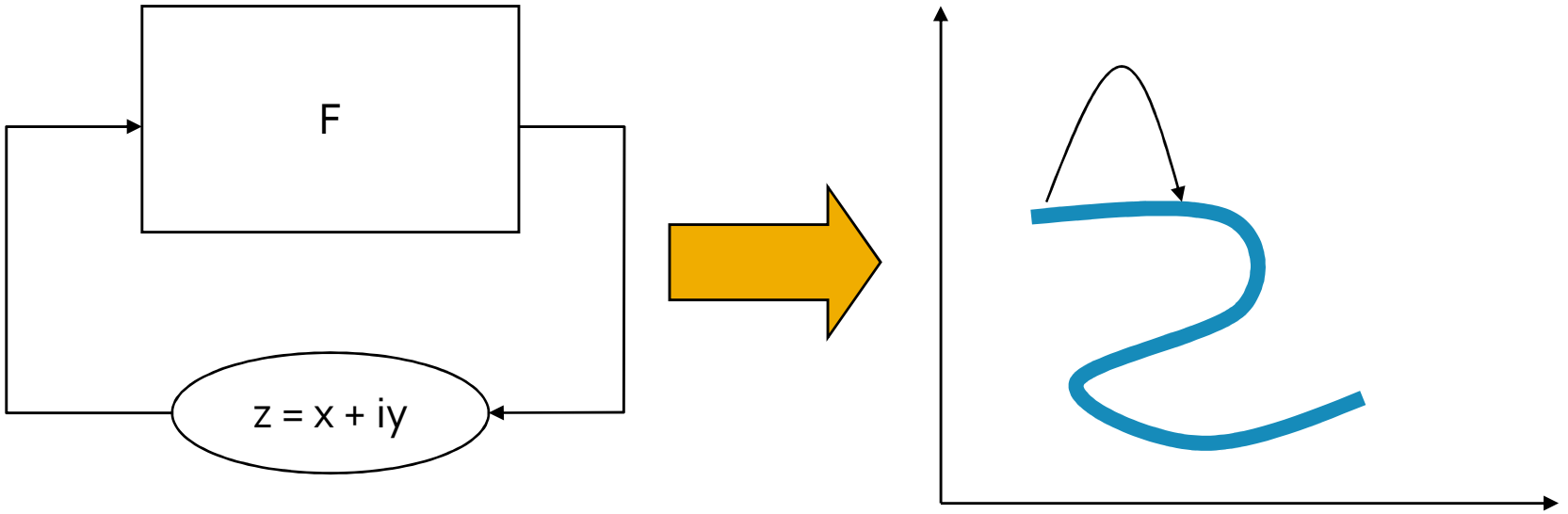
- Vertex shader

```
in vec4 position;                  // bemenő pozíció
in vec2 texCoord;                  // bemenő textúra koordináták
out vec2 fTexCoord;                // interpolálandó textúra koordináták

void main (void) {
    gl_Position = position;        // pozíció a pipeline további részére
    fTexCoord = texCoord;          // textúra koordináta a fragmens shadernek
}
```

- Fragmens shader

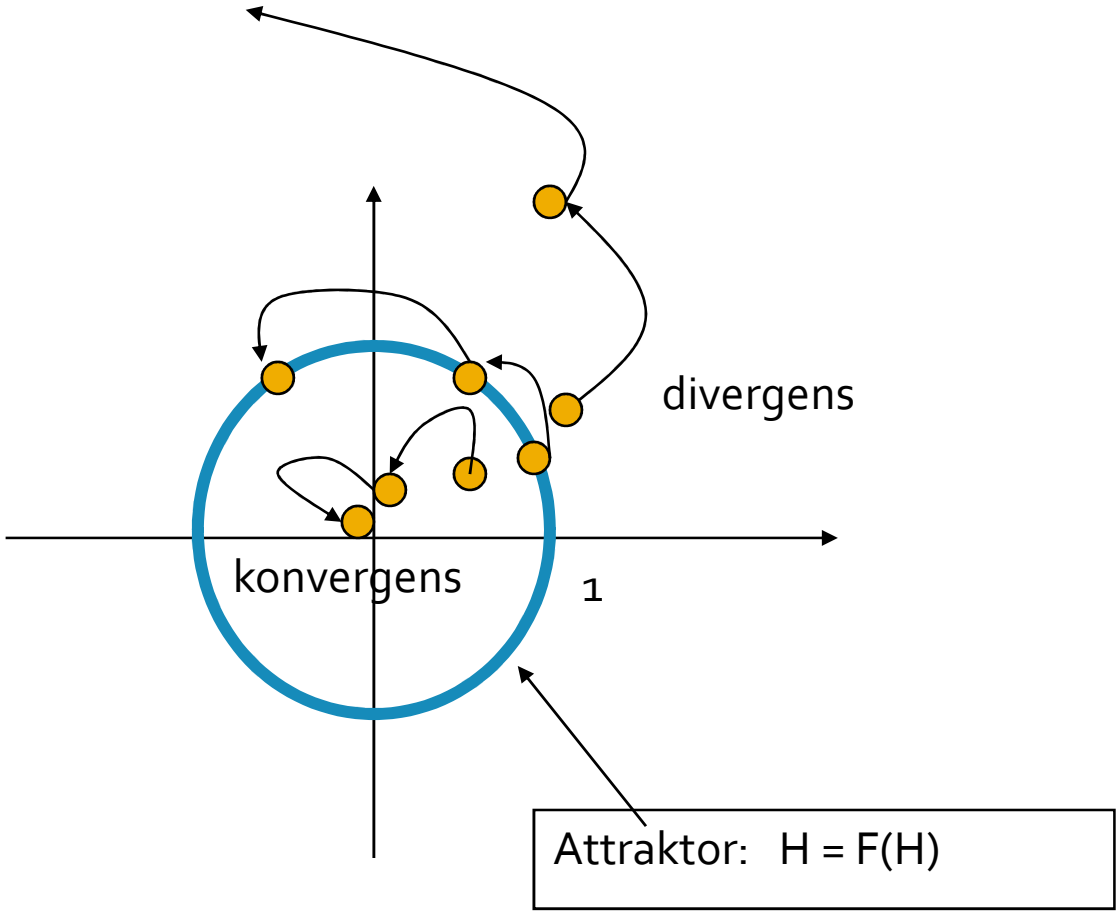
```
in vec2 fTexCoord;                 // interpolált textúra koordináta
out vec4 outColor;                 // a rasztertárba írandó szín

void main (void) {
    outColor = vec4(fTexCoord, 0.0, 1.0);
}
```

Példa vektor feldolgozásra: Iterált függvények attraktorai



- Egy pontba konvergál
- Divergál
- Egy tartományon ugrándozik: Attraktor

$z \rightarrow z^2$



$$z = r\, e^{i\phi}$$

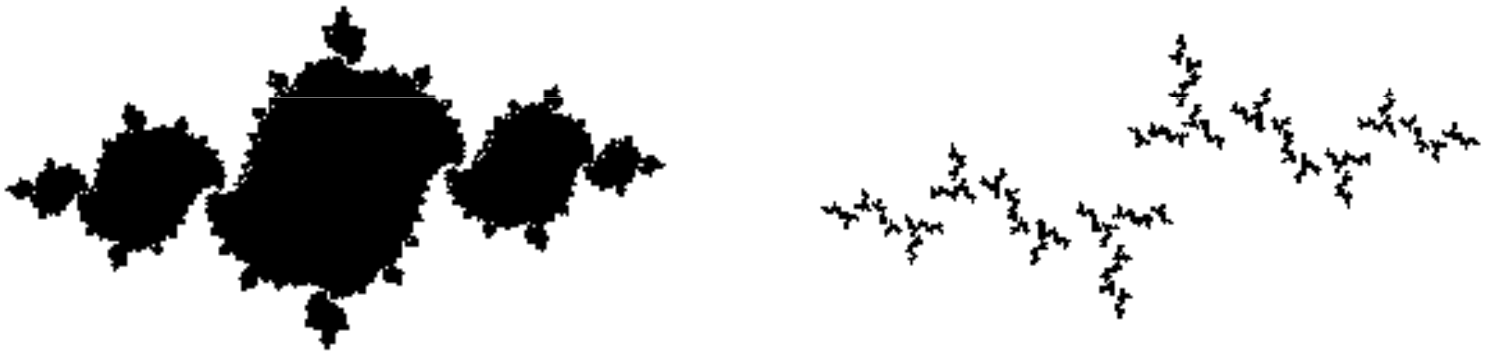
$$r \rightarrow r^2$$

$$\phi \rightarrow 2\phi$$

Attraktor előállítása

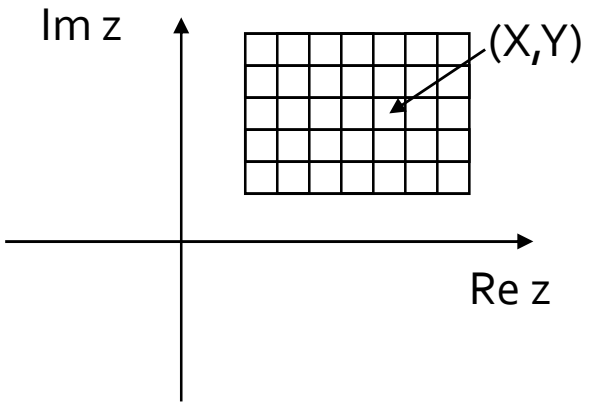
- Attraktor a labilis és a stabil tartomány határa
- Kitöltött attraktor = amely nem divergens
  - $z_{n+1} = z_n^2$: ha $|z_\infty| < \infty$ akkor fekete

Julia halmaz: $z \to z^2 + c$



Kitöltött Julia halmaz: algoritmus

```
FilledJuliaDraw ( )
   FOR Y = 0 TO Ymax DO
          FOR X = 0 TO Xmax DO
              ViewportWindow(X,Y → x, y)
              z = x + j y
              FOR i = 0 TO n DO z = z² + c
              IF |z| > "infinity" THEN  WRITE(X,Y, white)
              ELSE                      WRITE(X,Y, black)
          ENDFOR
   ENDFOR
END
```



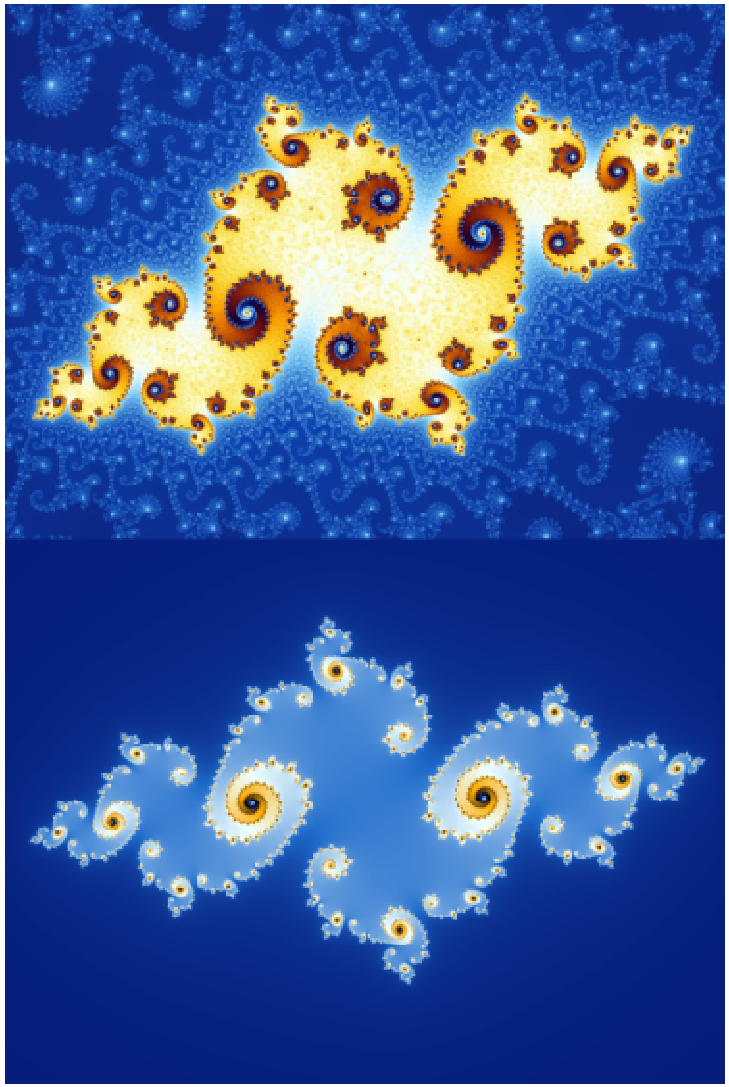
Vektor feldolgozás

- Julia halmaz

$$z_0 = x + iy$$

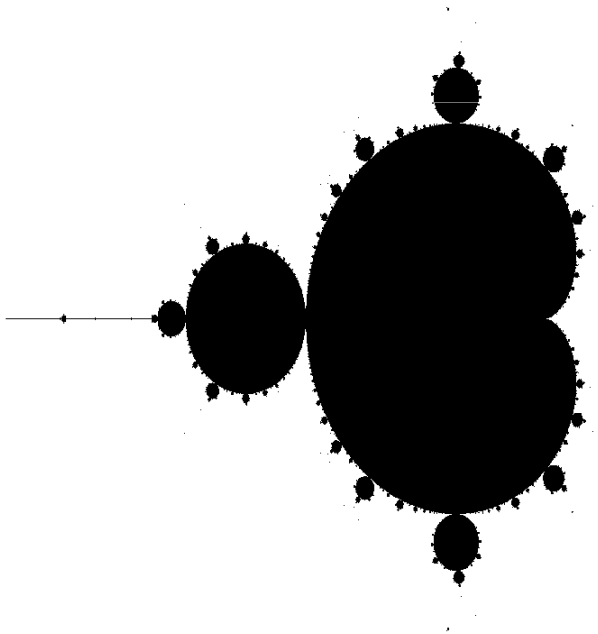
$$z_n = z_{n-1}^2 + c$$ Nem divergens

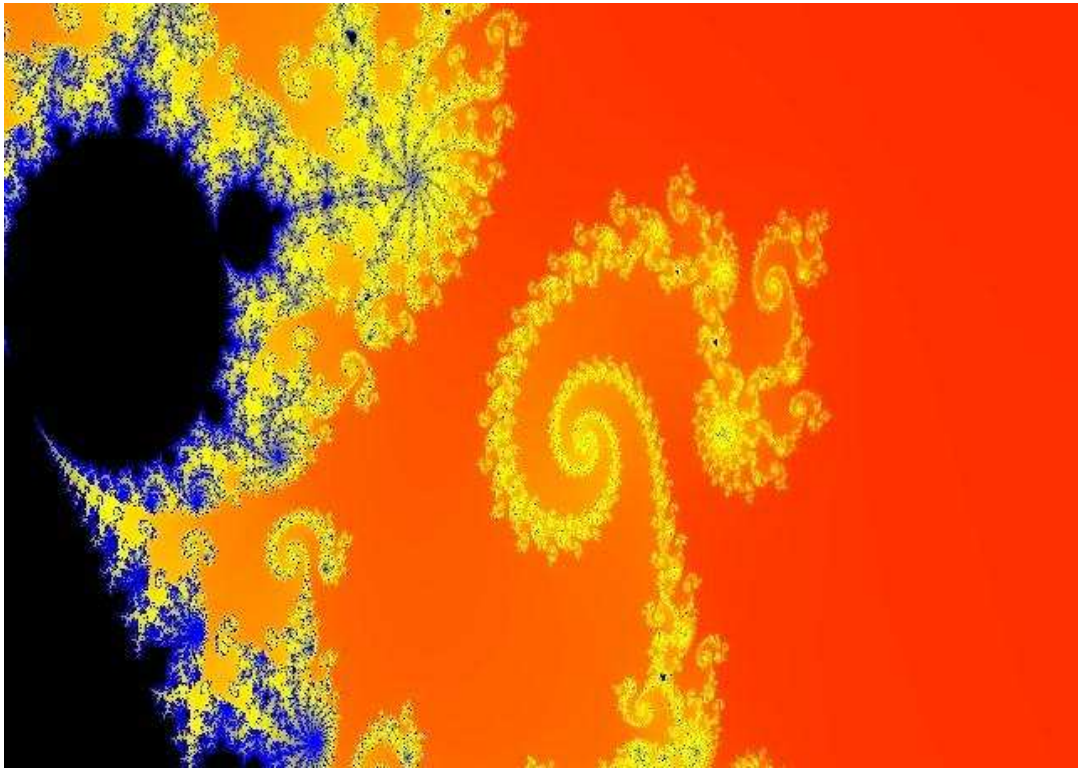
```
vec2 z = TexCoord;
int i;

for(i=0;i<maxIteration; ++i){
  z = vec2(z.x*z.x-z.y*z.y, 2.0*z.x*z.y)+c;
  if(dot(z, z) > 4.0){
    outColor = vec4(1.0);
    return;
  }
}
outColor = vec4(vec3(0.0), 1.0);
```



Mandelbrot halmaz

Azon c komplex számok, amelyekre a $z \rightarrow z^2 + c$ Julia halmaza összefüggő





Mandelbrot halmaz, algoritmus

```
MandelbrotDraw ( )
   FOR Y = 0 TO Ymax DO
          FOR X = 0 TO Xmax DO
              ViewportWindow(X,Y → x, y)
              c = x + j y
              z = 0
              FOR i = 0 TO n DO z = z² + c
              IF |z| > "infinity" THEN  WRITE(X,Y, white)
              ELSE                      WRITE(X,Y, black)
          ENDFOR
   ENDFOR
END
```

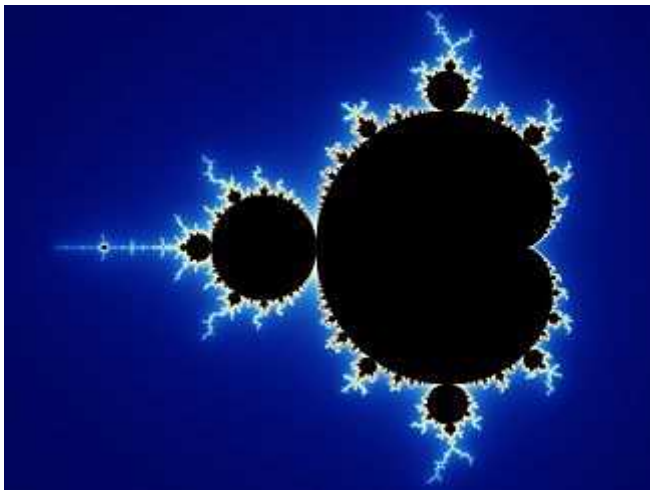
Vektor feldolgozás

- Mandelbrot halmaz

$$c = x + iy$$

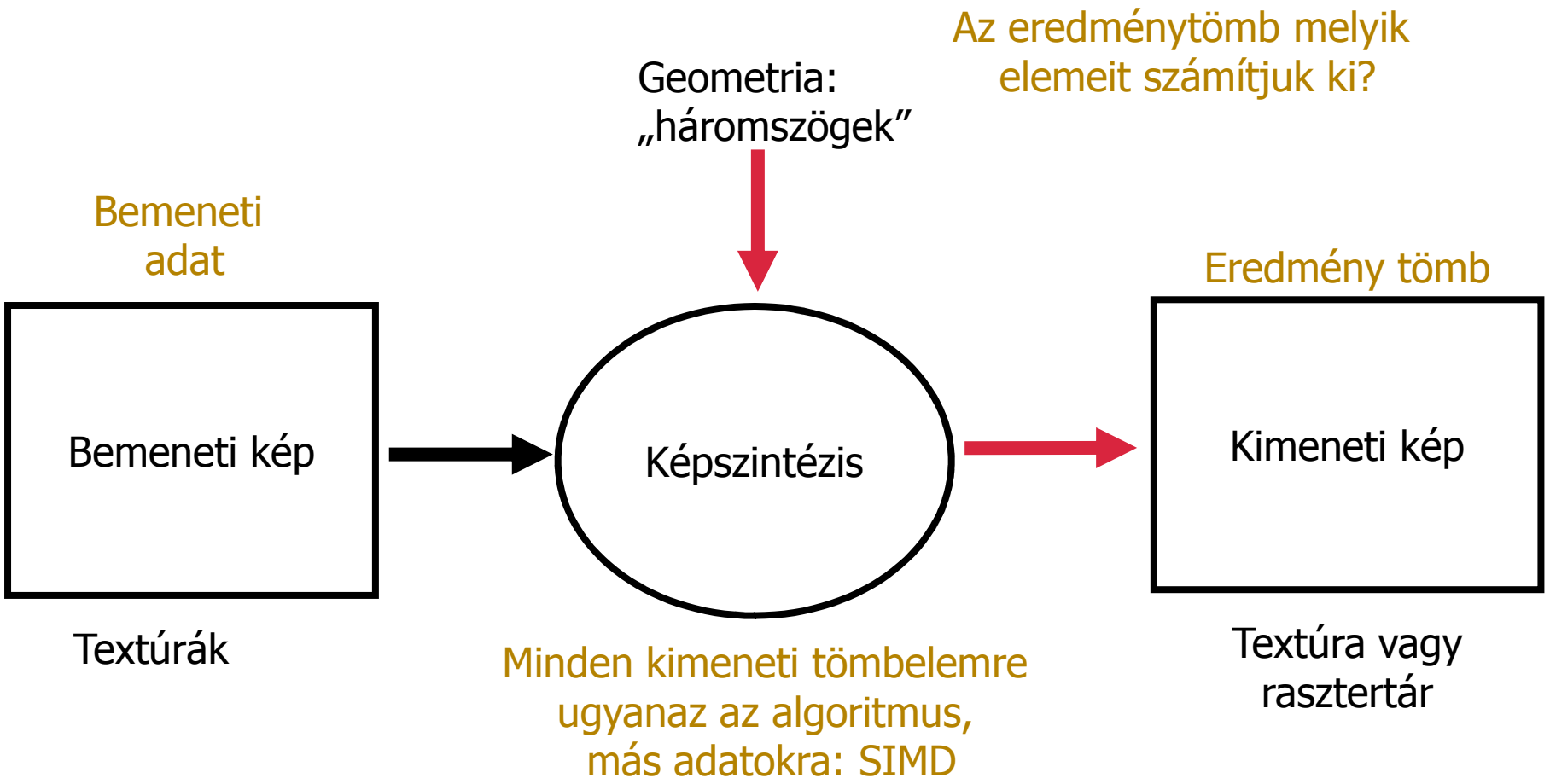
$$z_0 = c$$

$$z_n = z_{n-1}^2 + c$$ Nem divergens



```
vec2 c = TexCoord;
vec2 z = c;
int i;

for(i=0;i<maxIteration; ++i){
  z = vec2(z.x*z.x-z.y*z.y, 2.0*z.x*z.y)+c;
  if(dot(z, z) > 4.0){
    outColor = vec4(1.0);
  }
}
outColor = vec4(vec3(0.0), 1.0);
```

GPGPU: GPU mint vektorprocesszor



GPGPU: ocesszor

„Teljes képernyős" téglalap (CPU):

```
glViewport(0, 0, HRES, VRES)
glBegin(GL_QUADS);
glTexCoord2f(1,1); glVertex4f(-1,-1, 0, 1);
glTexCoord2f(1,0); glVertex4f(-1, 1, 0, 1);
glTexCoord2f(0,0); glVertex4f( 1, 1, 0, 1);
glTexCoord2f(0,1); glVertex4f( 1,-1, 0, 1);
glEnd( );
```

Vertex shader (GLSL):

```
in vec4 position;
in vec2 texCoord;
out vec2 fTexCoord;

void main (void) {
    gl_Position = position;
    fTexCoord = texCoord;
}
```

Fragment shader (GLSL):

```
uniform sampler2D bemAdat;
in vec2 fTexCoord;
out vec4 outColor;

void main (void) {
    outColor = Bemeneti képből számított adat a
               tex2D(bemAdat, f(Tex)) alapján;
}
```

Bemeneti adat

Melyik kimeneti tömbelemet számítjuk

Eredmény tömb

OpenGL: Textúrák

- Textúra definíció

```
GLuint texture;

glGenTextures(1, &texture);
glBindTexture(GL_TEXTURE_2D, texture);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F, w, h, 0, GL_RGBA, GL_FLOAT, 0);
```

- Shader paraméter

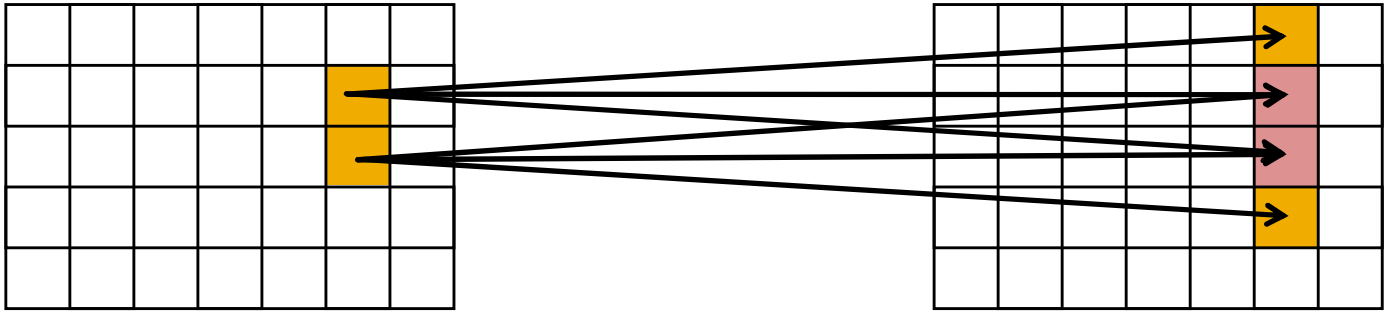
```
GLuint location = glGetUniformLocation(shaderProgram, „textureMap”);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture);
glUniform1i(location, 0);
```

- Shaderbeli elérés

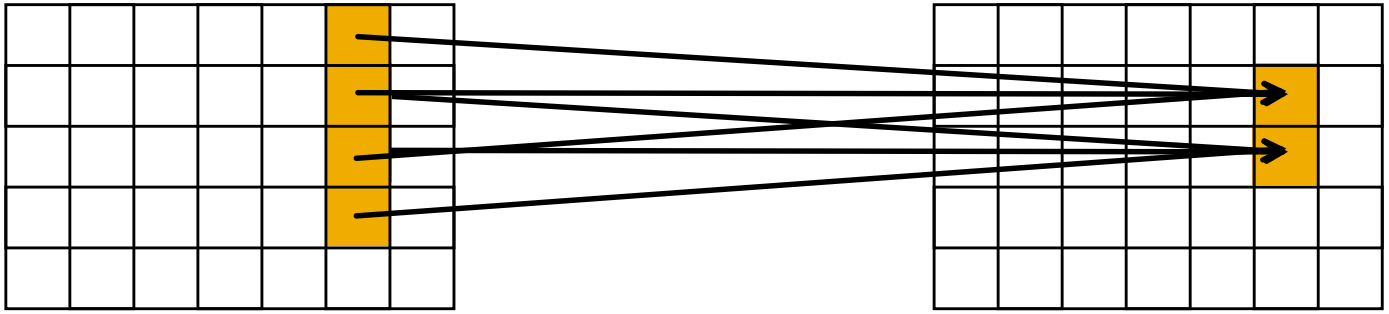
```
uniform sampler2D textureMap;

vec4 texColor = texture(textureMap, texCoord);
```

Párhuzamos sémák

- Szórás (Scatter)



- Gyűjtés (Gather)



Példa vektor feldolgozásra: Képfeldolgozás

- Fényességi transzformációk
  - CIE Luminancia

$$I = \begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} 0.21 \\ 0.39 \\ 0.4 \end{bmatrix}$$

```
uniform sampler2D colorMap;

in vec2 texCoord;
out vec4 outColor;

void main{
  vec4 color = texture(colorMap, texCoord);
  outColor =vec4(dot(color.rgb,
                     vec3(0.21, 0.39, 0.4)));
}
```

Képfeldolgozás

- Küszöbözés

```
uniform sampler2D colorMap;
uniform float threshold;

in vec2 texCoord;

float I(vec2 texCoord){
  vec4 color = texture(colorMap, texCoord);
  return(dot(color.rgb, vec3(0.21, 0.39, 0.4)));
}

void main(){
 outColor = I(texCoord) > threshold ?
                              vec4(1.0) : vec4(0.0);
}
```

Képszűrés

- Adatgyűjtés (Gather)
  - Egy-egy elem módosítása a környezete alapján
  - Konvolúciós szűrés (lehetne bármi más is)
    - Kép $\longrightarrow$ Kép transzformáció: $g_1 = f * g_0$ konvolúció

$$(f * g)(x,y) = \int_{u=-\infty}^{\infty} \int_{v=-\infty}^{\infty} f(u,v) \cdot g(x-u, y-v) du dv$$

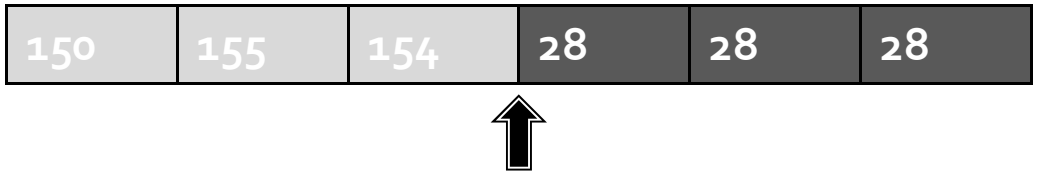
$$(f * g)(x,y) \approx \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} f(u,v) \cdot g(x-u, y-v) \Delta u \Delta v \qquad \Delta u \Delta v = 1$$

$$(f * g)(x,y) \approx \sum_{u=-1}^{1} \sum_{v=-1}^{1} f(u,v) \cdot g(x-u, y-v)$$

    - Képfeldolgozási műveletek

Képszűrés

- Élkeresés



Gradiens alapján detektáljuk az éleket

Central differences

$$\frac{\partial L}{\partial x}(x,y) = \frac{L(x+1,y) - L(x-1,y)}{2}$$

$$\frac{\partial L}{\partial y}(x,y) = \frac{L(x,y+1) - L(x,y-1)}{2}$$

$$I = \sqrt{\left(\frac{\partial L}{\partial x}\right)^2 + \left(\frac{\partial L}{\partial y}\right)^2}$$

Képszűrés

- Élkeresés

```
uniform sampler2D colorMap;
uniform vec2 textureSize;

in vec2 fTexCoord;
out vec4 outColor;

void main(){
 float gradX = (I(vec2(fTexCoord
                I(vec2(fTexCoord

 float gradY = (I(vec2(fTexCoord
                I(vec2(fTexCoord

 outColor = vec4(sqrt(gradX * gr

}
```

Képszűrés

- Élkeresés
  - Prewitt operátor

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \cdot I \qquad G_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \cdot I$$

  - Sobel operátor

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \cdot I \qquad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \cdot I$$

Képszűrés

- Prewitt operátor

```
uniform sampler2D colorMap;
uniform vec2 textureSize;
const float kernelX[9]=float[9]( -1.0/6.0, 0.0, 1.0/6.0,
                                 -1.0/6.0, 0.0, 1.0/6.0,
                                 -1.0/6.0, 0.0, 1.0/6.0 );
const float kernelY[9]=float[9]( 1.0/6.0,  1.0/6.0,  1.0/6.0,
                                     0.0,      0.0,      0.0,
                                -1.0/6.0, -1.0/6.0, -1.0/6.0 );
in vec2 fTexCoord;
out vec4 outColor;

void main(){
 float gradX = 0.0;
 float gradY = 0.0;

 for(int i=0; i<9; ++i){
   gradX += I(fTexCoord.xy + offset[i]) * kernelX[i];
   gradY += I(fTexCoord.xy + offset[i]) * kernelY[i];
 }

 outColor = vec4(sqrt(gradX * gradX + gradY * gradY));
}
```

Képszűrés

- Sobel operátor

```
uniform sampler2D colorMap;
uniform vec2 textureSize;
const float kernelX[9]=float[9]( -1.0/8.0, 0.0, 1.0/8.0,
                                 -2.0/8.0, 0.0, 2.0/8.0,
                                 -1.0/8.0, 0.0, 1.0/8.0 );
const float kernelY[9]=float[9]( 1.0/8.0,  2.0/8.0,  1.0/8.0,
                                     0.0,      0.0,      0.0,
                                -1.0/8.0, -2.0/8.0, -1.0/8.0 );
in vec2 fTexCoord;
out vec4 outColor;

void main(){
 float gradX = 0.0;
 float gradY = 0.0;

 for(int i=0; i<9; ++i){
   gradX += I(fTexCoord.xy + offset[i]) * kernelX[i];
   gradY += I(fTexCoord.xy + offset[i]) * kernelY[i];
 }

 outColor = vec4(sqrt(gradX * gradX + gradY * gradY));
}
```

Képszűrés

- Élkeresés
  - Laplace operátor

$$G_x = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \cdot I$$

```
uniform sampler2D colorMap;
uniform vec2 textureSize;
in vec2 texCoord;

const float kernel[9] = float[9]( 0.0, 1.0, 0.0,
                                  1.0, -4.0, 1.0,
                                  0.0, 1.0, 0.0);
void main(){
 for(int x=-1; x<1; ++x)
  for(int y=-1; y<1; ++y)
   outColor += I(texCoord + vec2(x/textureSize.x,
                                 y/textureSize.y))*
               kernel[x+y*3];
}
```

Képszűrés

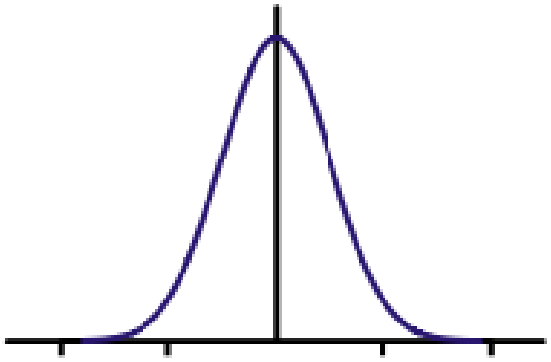
- Élkiemelés
  - A képből vonjuk ki a második deriváltját

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Képszűrés

- Gauss szűrő

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} /16$$