Scatter és gather típusú algoritmusok

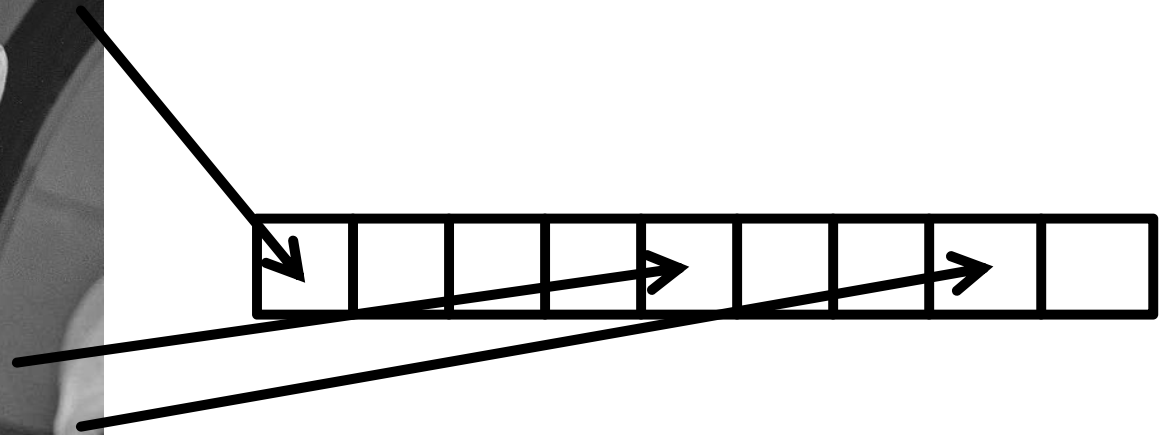
Hisztogram

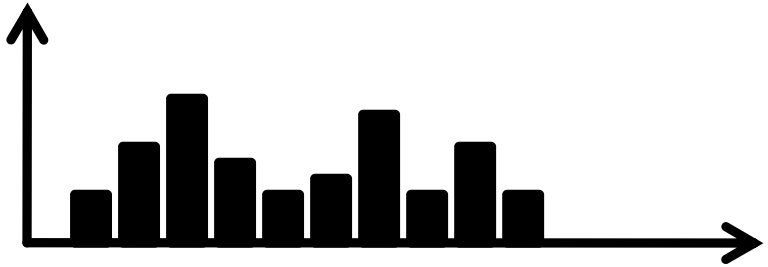
- Kvantált kép fényesség értékei: G [0, Gmax]
- G fényességű pontok száma: P(G)

$$p(g)\cdot\Delta g \qquad [g, g+\Delta g]$$

$$\int_0^{g_{\max}} p(g)\cdot dg = 1$$

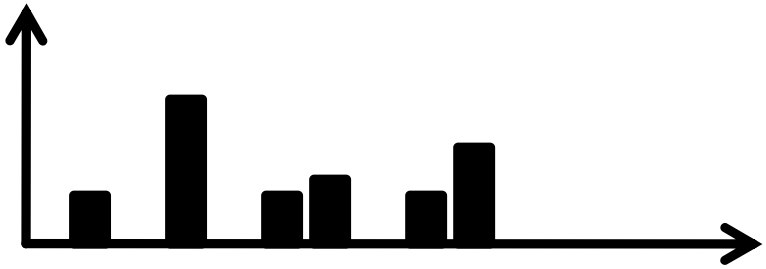
Hisztogram



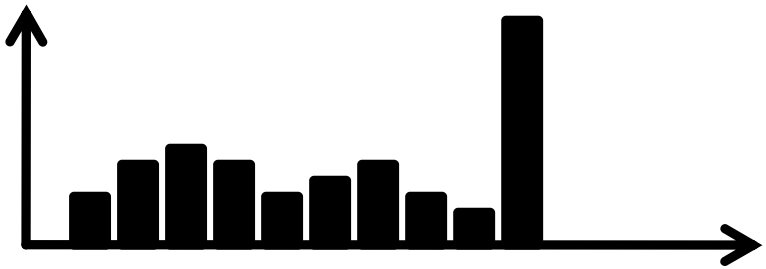


Hisztogram

- Kevés árnyalat

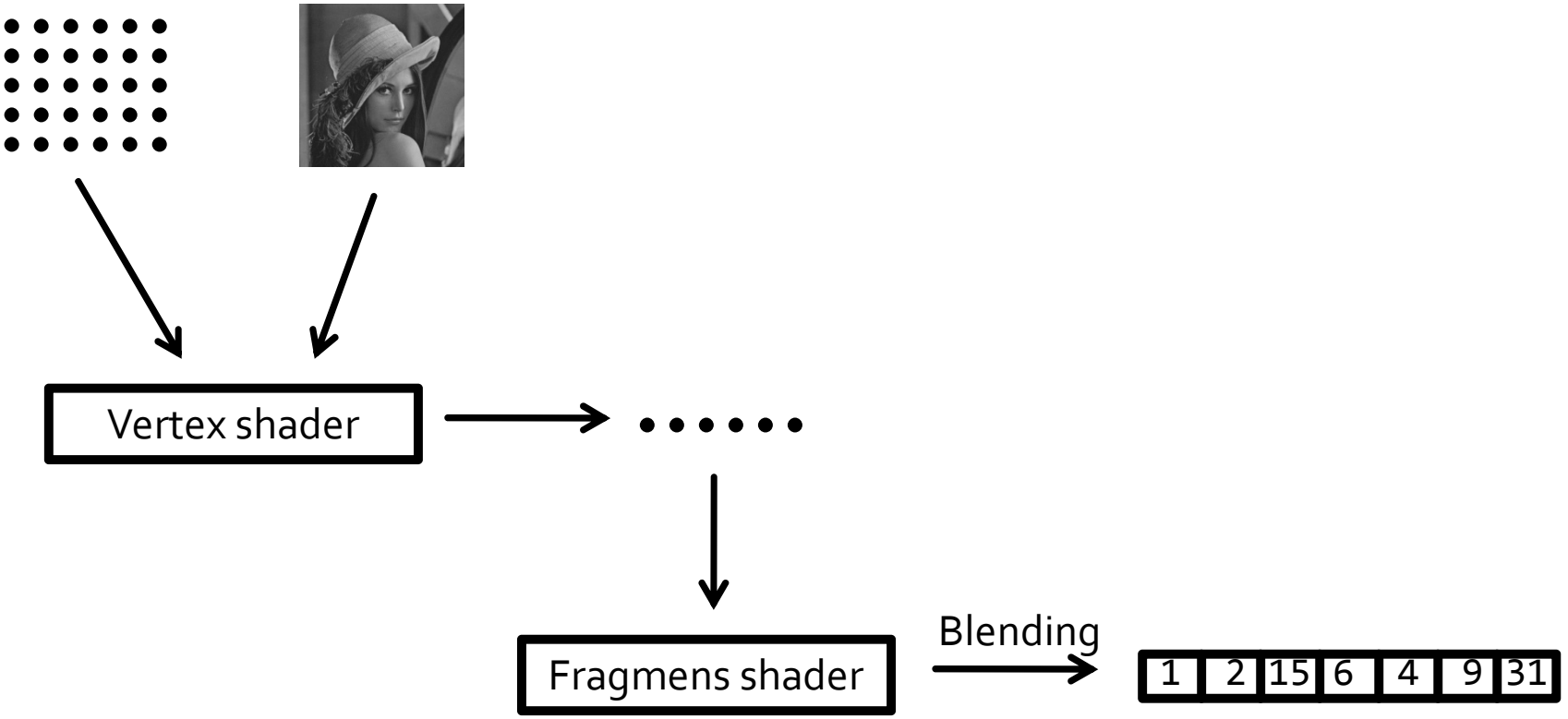


- Túlexponált



Hisztogram

- Hisztogram generálás a GPU-n



Hisztogram

- Fragmensek kompozitálása
  - glBlendEquation(equation)
    - GL_FUNC_ADD

$$R = R_s \cdot S_r + R_d \cdot D_r \qquad A = A_s \cdot S_a + A_d \cdot D_a$$

    - GL_FUNC_SUBTRACT

$$R = R_s \cdot S_r - R_d \cdot D_r \qquad A = A_s \cdot S_a - A_d \cdot D_a$$

    - GL_FUNC_REVERSE_SUBTRACT

$$R = R_d \cdot D_r - R_s \cdot S_r \qquad A = A_d \cdot D_a - A_s \cdot S_a$$

    - GL_MIN $R = \min(R_s, R_d) \qquad A = \min(A_s, A_d)$

    - GL_MAX $R = \max(R_s, R_d) \qquad A = \max(A_s, A_d)$

Hisztogram

- glBlendFunc(src, dst)
  - GL_ZERO, GL_ONE $S_r = 0 \quad A = 0$
  - GL_SRC_COLOR, GL_DST_COLOR $S_r = R_s \quad A = A_s$
  - GL_SRC_ALPHA, GL_DST_ALPHA $S_r = A_s \quad A = A_s$
  - GL_CONSTANT_COLOR, GL_CONSTANT_ALPHA $S_r = R_c \quad A = A_c$
  - GL_ONE_MINUS_... $S_r = 1 - R_s \quad A = 1 - A_s$

Hisztogram generálás

- Grid

```
GLfloat* vertices = new GLfloat[3 * width * height];
for(int y = 0; y < height; ++y){
 for(int x = 0; x < width; ++x){
  vertices[3 * (x + y * width)    ] = x / width;
  vertices[3 * (x + y * width) + 1] = y / height;
  vertices[3 * (x + y * width) + 2] = 0.0f;
 }
}
```

- Számítás

```
glEnable(GL_BLEND);
glBlendFunc(GL_ONE, GL_ONE);
histogramBuffer->setRenderTarget();
histogramShader->enable();
histogramShader->bindUniformTexture(„inputBuffer”, texture->getTextureHandle(0), 0);
grid->render(histogramShader);
histogramShader->disable();
histogramBuffer->disableRenderTarget();
glDisable(GL_BLEND);
```

Hisztogram generálás

- Vertex shader

```
uniform sampler2D inputBuffer;
in vec4 position;

void main(void){
 vec2 resolution = textureSize(inputBuffer, 0);
 float luminance = texture(inputBuffer, position.xy + (0.5 / resolution)).x;
 gl_Position = vec4( 2.0 * (luminance - 0.5), 0.0, 0.0, 1.0);
}
```

- Fragmens shader

```
out vec4 outColor;

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

Hisztogram generálás

- Vertex shader II.

```
uniform sampler2D inputBuffer;
uniform float hLevels;

in vec4 position;

void main(void){
 vec2 resolution = textureSize(inputBuffer, 0);
 float luminance = texture(inputBuffer, position.xy + (0.5 / resolution)).x;
 gl_Position = vec4( 2.0 * (luminance * (1.0 – 1.0 / hLevels) + 0.5 / hLevels - 0.5),
                     0.0, 0.0, 1.0);
}
```

Kép visszaolvasása

- Forrás kiválasztása
  - Buffer kiválasztása

```
glBindBuffer(target, buffer);
```

    - GL_DRAW_FRAMEBUFFER
    - GL_READ_FRAMEBUFFER
    - GL_FRAMEBUFFER
  - Komponens kiválasztása

```
glReadBuffer(source);
```

    - GL_COLOR_ATTACHMENTx
    - GL_DEPTH_COMPONENT
    - GL_DEPTH_STENCIL
    - GL_STENCIL_INDEX

Kép visszaolvasása

- Buffer másolása

```
glReadPixels(x, y, width, height, format, type, *data);
```

  - Format
    - GL_RED, GL_GREEN, GL_BLUE, GL_ALPHA
    - GL_RGB, GL_RGBA
    - GL_LUMINANCE, GL_LUMINANCE_ALPHA
  - Type
    - GL_BYTE, GL_SHORT, GL_INT, GL_FLOAT
    - GL_UNSIGNED_INT_8_8_8_8
    - …

Kép visszaolvasása

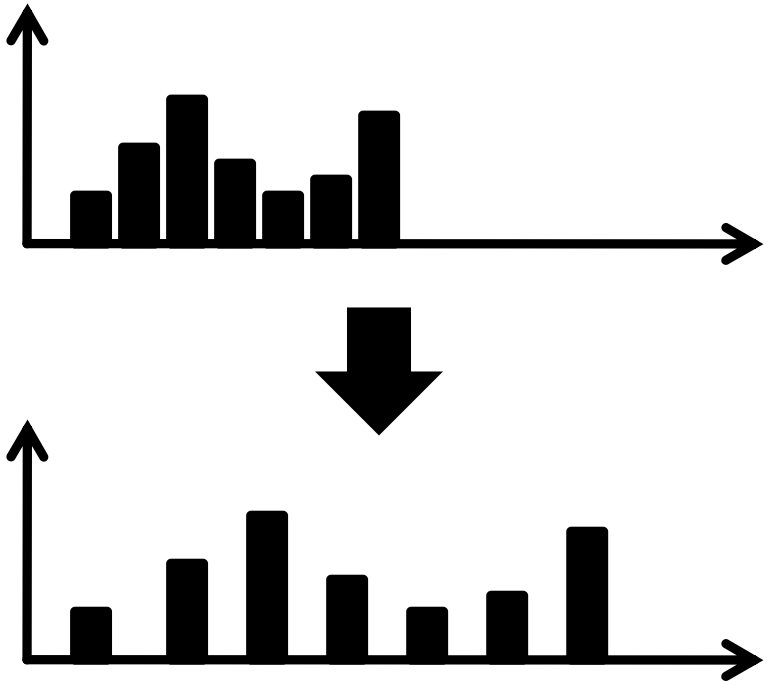
```
float histogram[255] = {0.0f};

glBindFramebuffer(GL_FRAMEBUFFER, histogramBuffer->getHandle());
glReadBuffer(GL_COLOR_ATTACHMENT0);
glReadPixels(0, 0, 255, 1, GL_RED, GL_FLOAT, histogram);
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

Hisztogram

- Normálás

$$G^* = \frac{G_{\max}}{G_2 - G_1}(G - G_1)$$



Hisztogram

- Normálás

```
uniform sampler2D inputMap;
uniform float G1;
uniform float G2;
uniform float Gmax;

in vec2 fTexCoord;
out vec4 outColor;

void main(void){
 float luminance = texture(inputMap, fTexCoord);

 outColor = vec4( Gmax / (G2 – G1) * (luminance – G1));
}
```

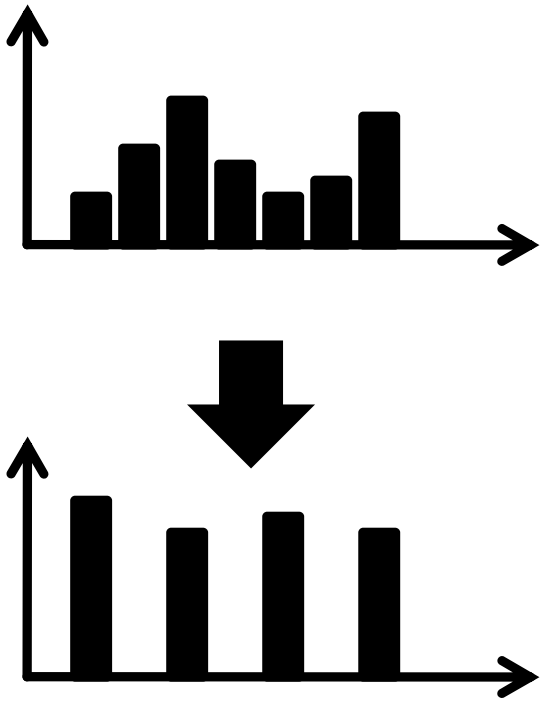
Hisztogram

- Kvantálás

```
uniform sampler2D inputMap;
uniform int levels;

in vec2 fTexCoord;
out vec4 outColor;

void main(void){
 float c = texture(inputMap, fTexCoord);
 float threshold = 1.0 / levels;
 while(c > threshold){
  threshold += 1.0 / levels;
 }
 outColor = vec4(threshold);
}
```



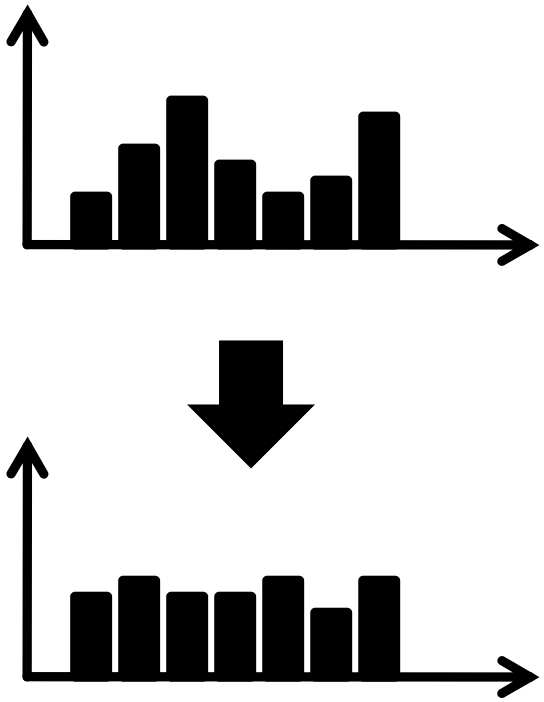
Histogram kiegyenlítés

$$p(g) \rightarrow h = f(g) \rightarrow p(h)$$

$$p(h) = p(g) \cdot \frac{dg}{dh}$$

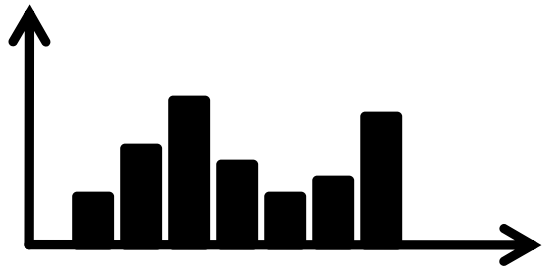
$$\frac{1}{h_{\max}} = p(g)\frac{dg}{dh} \Rightarrow \frac{dh}{dg} = h_{\max} p(g)$$

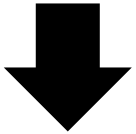
$$\boxed{h(g) = h_{\max} \int_0^g p(\gamma) d\gamma}$$



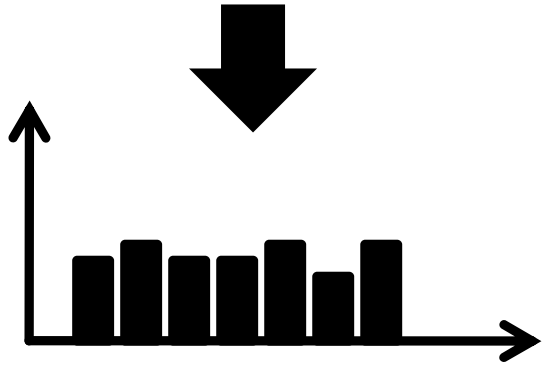
Histogram kiegyenlítés

$$N_{pix} = \sum_{G=0}^{G_{\max}} P(G) \qquad P(G) = \frac{P(G)}{N_{pix}}$$



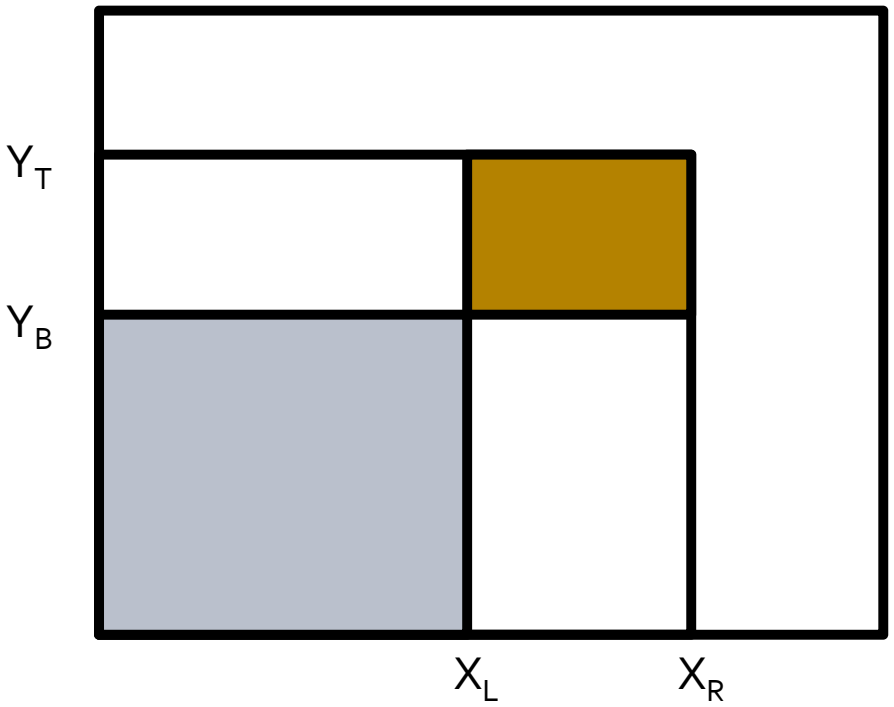


$$H = \text{int}\left(\frac{H_{\max}}{N_{pix}} \sum_{\Gamma=0}^{G} P(\Gamma) \right)$$



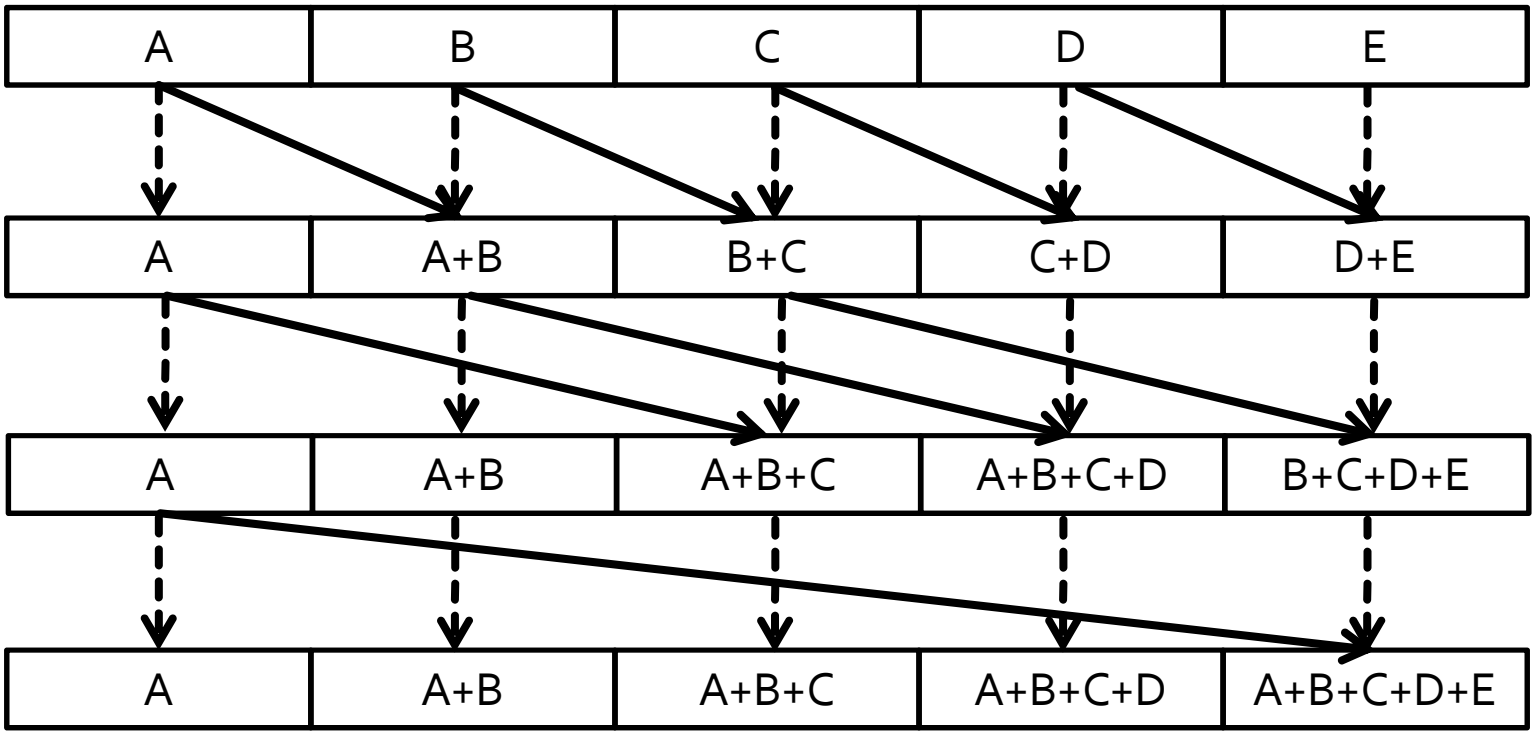
Histogram kiegyenlítés

- Summed area table
  - Mip-map alternatíva
  - Minden elemben egy összeg van
  - Gyorsan számítható
  - $\log_2(n)$ időben generálható



Histogram kiegyenlítés

- SAT generálás



Histogram kiegyenlítés

- SAT generálás

```
tA = input image
n = logr(width)
m = logr(height)

// horizontal phase
for(i=0; i<n; i=i+1)
 tB[x,y] = tA[x,y] +
           tA[x+1*ri, y] +
           tA[x+2*ri, y] +
           ...
           tA[x+r*ri, y]

 swap(tA, tB)
```

Histogram kiegyenlítés

- SAT generálás
  - OpenGL

```
float offset = 1.0f / (float)histogramLevels;
int inputBuffer = 0;
for(int i=0; i<8; ++i){
 sat[(inputBuffer + 1) % 2]->setRenderTarget(0);
 satShader->enable();
 satShader->bindUniformTexture(„inputMap”, sat[inputBuffer]->getColorBuffer(0), 0);
 satShader->bindUniformFloat(„offset”, -offset);
 fullscreenQuad->render(satShader);
 sat[(inputBuffer + 1) % 2]->disableRenderTarget();

 inputBuffer = (inputBuffer + 1) % 2;
 offset *= 2.0f;
}
```

Histogram kiegyenlítés

- SAT generálás
  - Fragmens shader

```
uniform sampler2D inputMap;
uniform float offset;

in vec2 fTexCoord;
out vec4 outColor;

void main(void){
 float current = texture(inputMap, fTexCoord).x;
 float sampleOff = fTexCoord.x + offset;
 float addValue = 0.0;

 if(sampleOff >= 0.0){
  addValue = texture(inputMap, vec2(sampleOff, 0.0)).x;
 }

 outColor = vec4(current + addValue);
}
```

Histogram kiegyenlítés

- Kiegyenlítő leképzés

```
uniform sampler2D inputMap;
uniform sampler2D histogram;
uniform float level;

in vec2 fTexCoord;
out vec4 outColor;

void main(void){
 float current = texture(inputMap, fTexCoord).x;
 float accum = texture(histogram, vec2(current, 0.0)).x;

 float bin = floor(accum / level);

 outColor = vec4(bin);
}
```

Tone mapping

- Dinamika tartományok

  $[0,1] \rightarrow [0,\infty]$

- Leképzés HDR-ről LDR-re
  - Luminancia: Y
  - Átlagos luminancia: Y′

$$Y_r = \frac{\alpha Y}{Y'} \longrightarrow D = \frac{Y_r\left(1+\frac{Y_r}{W^2}\right)}{1+Y_r}$$

Tone mapping

Tone mapping

- A leképzés lépései

```
tA = eredeti kép

tL = luminancia(tA)

l_átlag = átlagol(tL)

tB = transform(tA, l_átlag)
```

  - Luminancia leképzés

```
float luminance(vec4 color){
 return vec4(color.r * 0.2126, color.g * 0.7152, color.b * 0.0722, 1.0);
}
```

Tone mapping

- Átlag képzés
  - Sat
  - Textúra piramis

Tone mapping

- Átlag képzés

```
uniform sampler2D inputBuffer;

in vec2 fTexCoord;
out vec4 outColor;

void main(void){
 vec2 offset = 1.0 / textureSize(inputBuffer, 0);
 float sum = 0.0;
 for(int y = -1; y <= 1; ++y){
  for(int x = -1; x <= 1; ++x){
   sum += texture(inputBuffer, fTexCoord + vec2(x, y) * offset);
  }
 }
 outColor = vec4(sum / 9.0);
}
```

Tone mapping

- Átlagos luminancia

```
uniform sampler2D inputBuffer;
uniform float Yavg;
uniform float alpha;

in vec2 fTexCoord;
out vec4 outColor;

void main(void){
 float Y = texture(inputBuffer, fTexCoord);
 outColor = vec4(alpha * Y / Yavg);
}
```

$$Y_r = \frac{\alpha Y}{Y'}$$

Tone mapping

- Végső transzformáció

```
uniform sampler2D inputBuffer;
uniform float W;

in vec2 fTexCoord;
out vec4 outColor;

void main(void){
 vec4 Yr = texture(inputBuffer, fTexCoord);
 outColor = vec4(Yr * (1 + Yr / (W * W)) / (1.0 + Yr));
}
```

$$D = \frac{Y_r\left(1+\frac{Y_r}{W^2}\right)}{1+Y_r}$$

Tone mapping