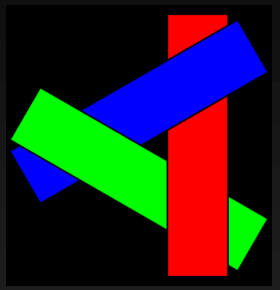
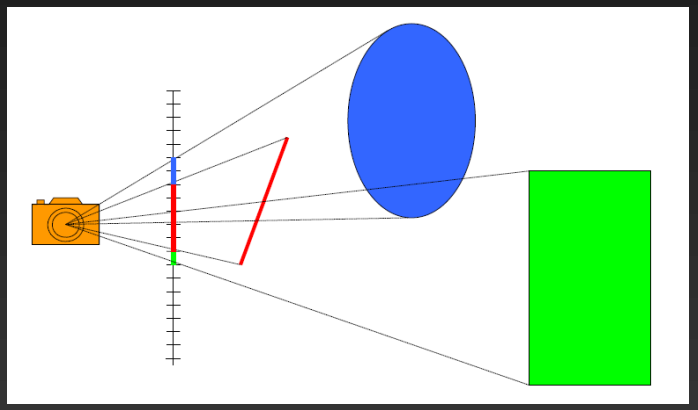
Průhlednost, mlha a zakrývání podle Z

Paměť hloubky

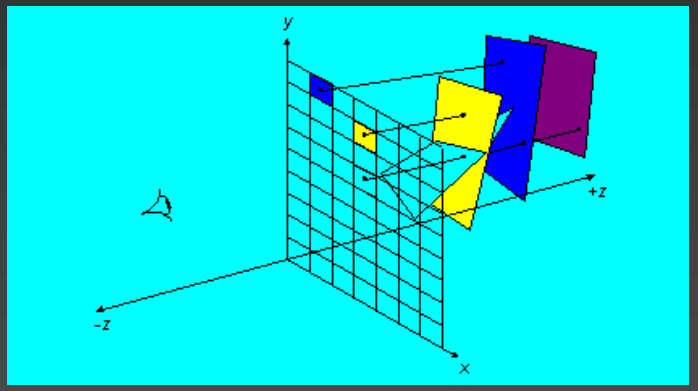
- Neboli Z-buffer (depth buffer)



- Nutná pro korektní zobrazení v situacích neřešitelných malířovým algoritmem



- Blokuje zápis fragmentů do výsledného obrázku
  - vzdálenější fragment je blokován



- Obvykle 1 Z-buffer na snímek

Použití paměti hloubky

- Without Z-Buffer (depth buffer) all polygons are displayed in draw order, no matter the distance and visibility
  - manually – painters algorithm – too hard for complex scenes
- For proper per fragment decision we need:
  1) enable depth test

     glEnable(GL_DEPTH_TEST)
  2) define „zero“ depth (usually not necessary)

     glClearDepth(depth)
     - from 0.0 to 1.0 (near plane to far plane)
  3) choose type of depth test

     glDepthFunc(func)

     GL_NEVER, GL_LESS, GL_EQUAL, GL_LEQUAL, GL_GREATER, GL_NOTEQUAL, GL_GEQUAL, GL_ALWAYS
     - most common GL_LESS or GL_LEQUAL (close hides distant)
  4) clear Z-buffer when starting new frame

     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

Průhlednost

- RGBA, RGB+alfa kanál - „neprůhlednost“, krytí

  A=0.0 plně průhledné
- Určuje způsob kombinace vykreslovaných fragmentů s color-bufferem
  - fragment = barva + průhlednost + Z-souřadnice...
- Bez míchání barva fragmentu přepíše existující hodnotu
- Výpočet nové barvy podle zvolené funkce
  - smícháním stávající a nově příchozí

Míchání (blending)

- Zdroj = příchozí fragment
- Cíl = obsah již uloženého pixelu v color-bufferu
- Mísicí rovnice

$$R_n = R_s \cdot S_r + R_d \cdot D_r$$
$$G_n = G_s \cdot S_g + G_d \cdot D_g$$
$$B_n = B_s \cdot S_b + B_d \cdot D_b$$
$$A_n = A_s \cdot S_a + A_d \cdot D_a$$

- RGBA, S(D) – míchací faktor zdroje (cíle)
- Indexy: rgb, n – nový pixel, s – source, d – dest.

- glBlendEquation(GLenum mode)

  GL_FUNC_ADD, GL_FUNC_SUBTRACT, GL_FUNC_REVERSE_SUBTRACT, GL_MIN, GL_MAX

Nastavení mísicí funkce

- Shodné faktory pro RGB i A

  glBlendFunc(srcfactor, dstfactor)
- Různé faktory pro RGB, A

  glBlendFuncSeparate(srcRGB, dstRGB, srcA, dstA)
- Faktory vybírány z tabulky (viz OpenGL dokumentace)
  - nejběžnější příklad:
    - zdrojový faktor: GL_SRC_ALPHA

      nastaví pro RGBA faktor (As,As,As,As)
    - cílový faktor: GL_ONE_MINUS_SRC_ALPHA

      nastaví pro RGBA faktor (1-As,1-As,1-As,1-As)
    - výsledek (pro červenou složku) Rd=As*Rs+(1-As)*Rd

Použití: jak vykreslit průhledná tělesa

- Nastavit faktory (někde v init, obvykle už se nemění)

  glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
- Každý snímek
  - Vykreslit všechna neprůhledná tělesa v libovolném pořadí
  - Pro průhledná tělesa
    - Povolit mísení

      glEnable(GL_BLEND)
    - Zablokovat paměť hloubky pouze pro čtení
      - těleso je průhledné, neblokovat vykreslování když je něco za ním, chceme to vidět

      glDepthMask(GL_FALSE)
    - Zablokovat zahazování zadních stěn polygonu (vidíme skrz)

      glDisable(GL_CULL_FACE)
    - Vykreslit průhledná tělesa (odzadu dopředu, nebo pomocí OIT)
    - Zablokovat mísení, povolit test hloubky, povolit zahazování

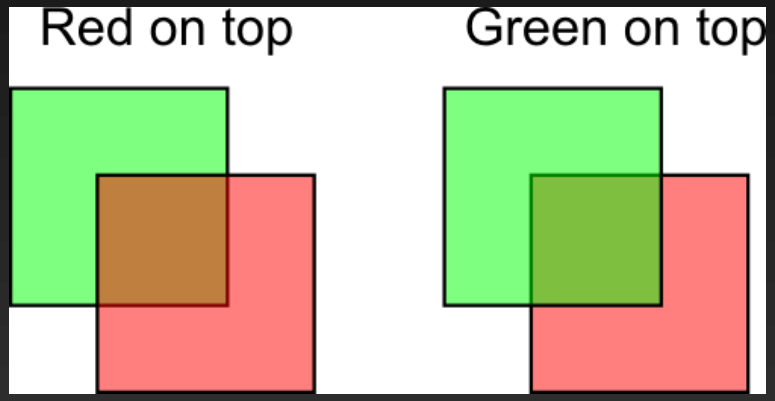
      glDisable(GL_BLEND)

      glDepthMask(GL_TRUE)

      glEnable(GL_CULL_FACE)

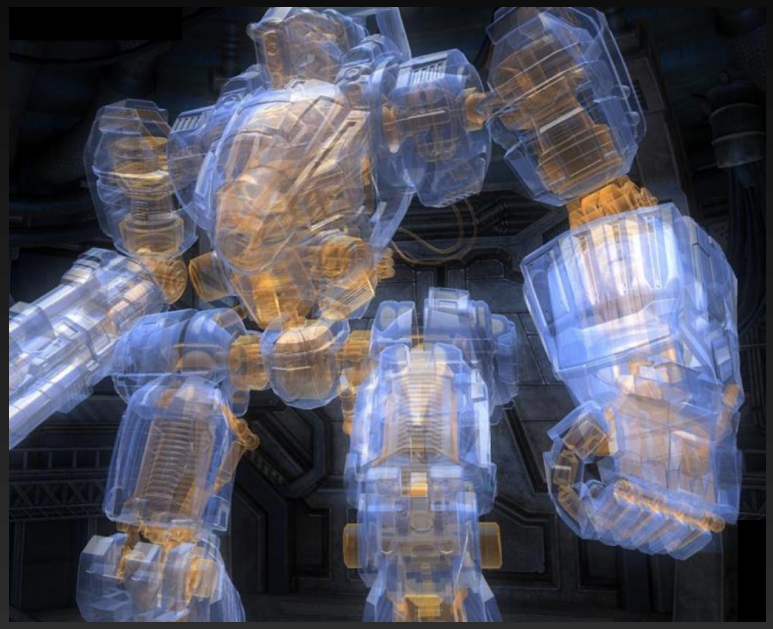
Závislost na pořadí operací

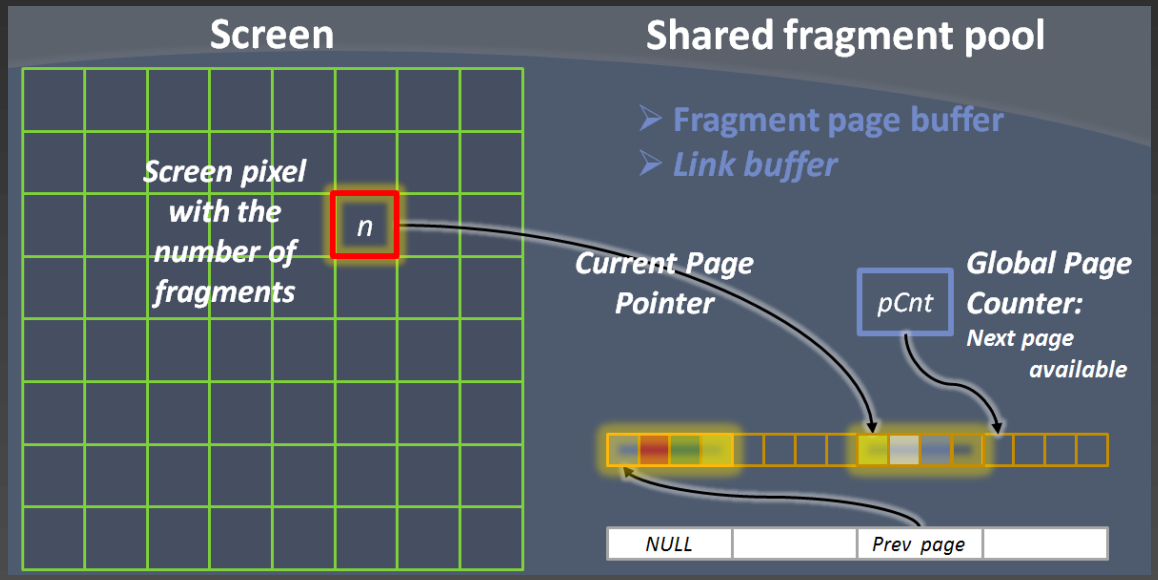
$$RGB_d = A_s \times RGB_s + (1.0 - A_s) \times RGB_d$$
$$RGB_d = RGB_d + A_s \times (RGB_s - RGB_d)$$

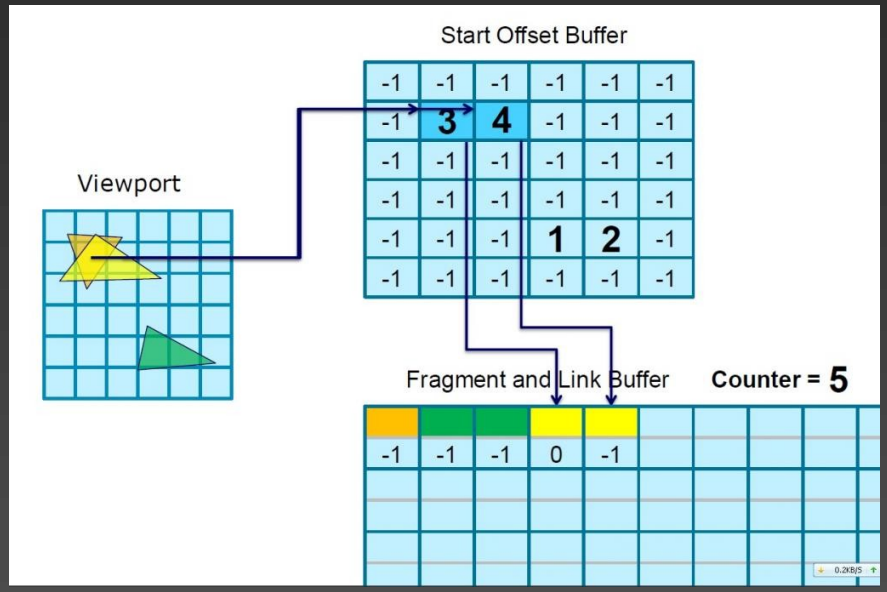


OIT
Order Independent Transparency

- A-Buffer, Depth peeling
- atomické čítače, HW 2011
  - OpenGL - 2012
  - D3D – 2015 (v.12)

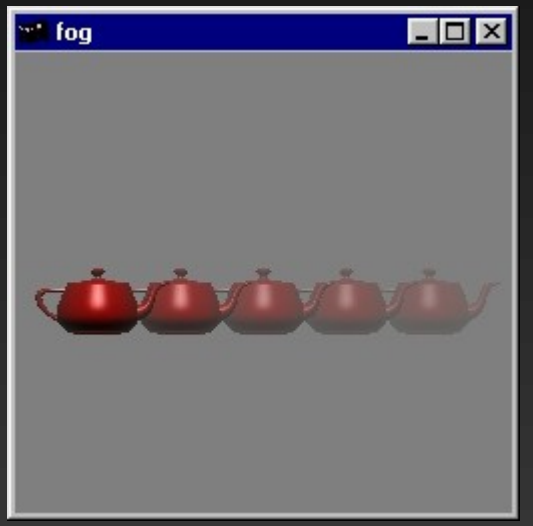






Mlha

- Hlavní důvody
  - zvýšení reálnosti
  - zvýšení rychlosti zobrazení
    - těsně za hustou mlhou nastavíme ořez
- Hustota se zvyšuje se vzdáleností
  - parametricky nastavitelné
  - různé matematické funkce mlhy
- Libovolná barva
  - obvykle šedá nebo černá
    (objekty mizí v dálce v oparu nebo ve tmě)



Možnosti mlhy

- Lineární

$$f = \frac{end - z}{end - start}$$

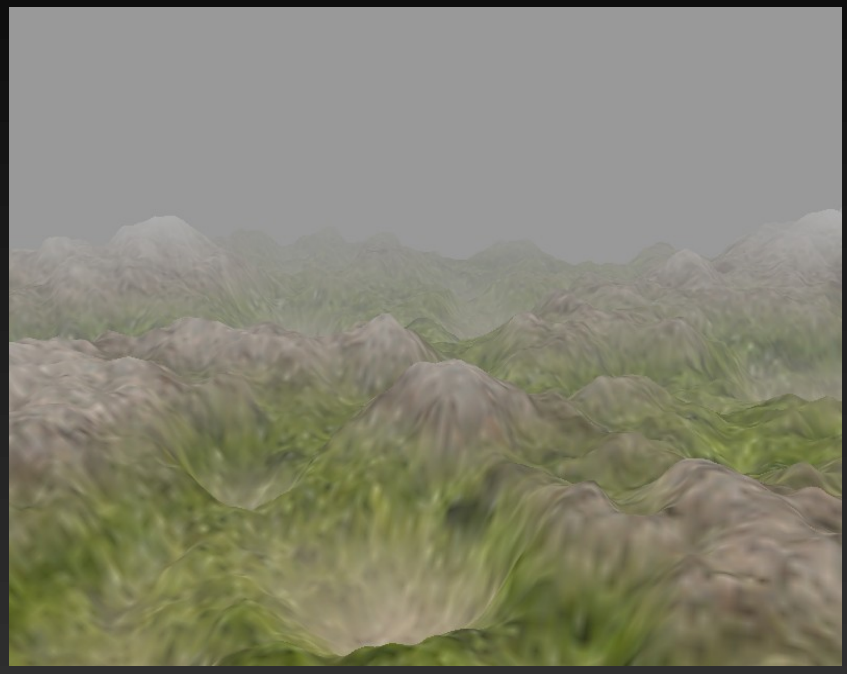
- Exponenciální

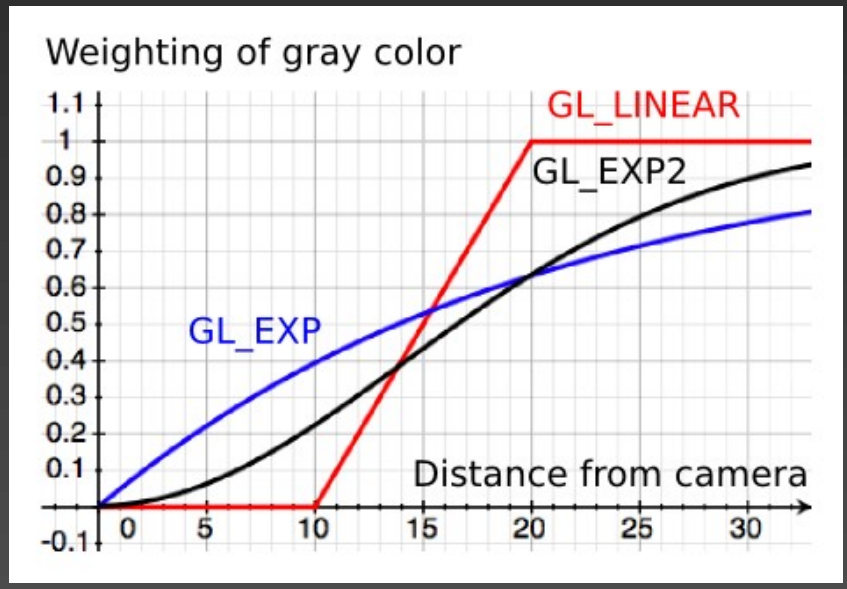
$$f = e^{-density \cdot z}$$

- Exponenciální kvadratická

$$f = e^{-(density \cdot z)^2}$$

  - nejlépe odpovídá realitě
  - kvadratický úbytek světla se vzdáleností





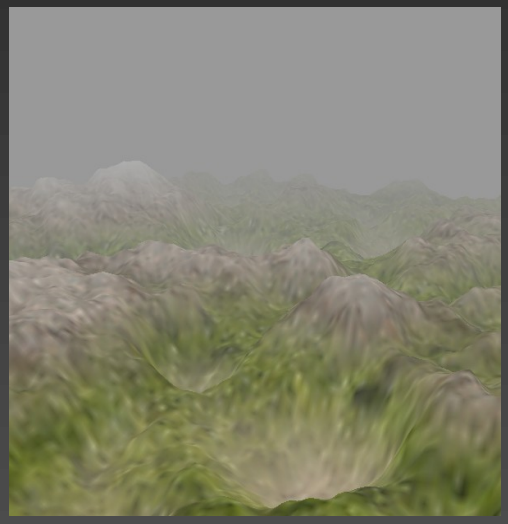
Fog in fragment shader

```
uniform vec4 fog_color = vec4(vec3(0.0f), 1.0f); // black, non-transparent = night
uniform float fog_density = 1.5f;

void main(void)
{
    vec4 color = ... lights ... textures ...

    float fog = exp(-fog_density * fog_density * gl_FragCoord.z * gl_FragCoord.z);
    fog = clamp(fog, 0.0, 1.0);

    // outputs final color
    FragColor = mix(color, vec4(fog_color, 1.0), 1-fog); //linear interpolation
}

// ====== or (for example) ======

uniform vec4 fog_color = vec4(vec3(0.0f), 1.0f); // black, non-transparent = night
uniform float near = 0.1f;
uniform float far  = 20.0f;

float log_depth(float depth, float steepness = 0.5f, float offset = 15.0f)
{
    float linear_depth = (2.0 * near * far) / (far + near - (depth * 2.0 - 1.0) * (far - near));
    return (1 / (1 + exp(-steepness * (linear_depth - offset))));
}

void main()
{
    vec4 color = ... lights ... textures ...

    // outputs final color
    float depth = log_depth(gl_FragCoord.z);
    FragColor = mix(color, vec4(fog_color, 1.0), depth); //linear interpolation
}
```

$$f = \mathrm{e}^{-(density \cdot z)^2}$$

Další vlastnosti jednoduché mlhy

- Per-fragment operace v závěru pipeline
- Závisí jen na vzdálenosti od kamery
  - Z souřadnice, nezávisí na výšce nad terénem apod.
  - nelze vyrobit přízemní mlhu, cáry mlhy
    - jen přes analýzu scény (a FS)



Shrnutí real-time 3D grafiky :-)