1)Vertex load and transformation

2)Rasterisation

3)Fragment coloring
- textures, materials, lights

4)Transparency and depth computation

Framebuffer

- Soubor bufferů nutných k vytvoření snímku
  - vertex buffer (VBO)
  - element buffer (EBO)
  - color buffer
  - depth buffer (Z-buffer)
  - stencil buffer
    - šablona
- Také
  - Frame buffer object (FBO)
    - Off-screen rendering
  - Pixel buffer object (PBO)
    - Rychlejší transfer textur
  - Uniform buffer object (UBO)
    - přenos mnoho uniform proměnných

Buffer

- Buffer = lineární paměť v prostoru GPU
  - identifikovaná jménem

    `Gluint`
- Workflow
  - Alokovat jméno (ID)

    `void glCreateBuffers(GLsizei n, GLuint *buffers)`
  - Získat data
    - Naplnit daty + očekávané využití

```
void glNamedBufferData(GLuint buffer, GLsizeiptr size, const void *data, GLenum usage)
```

      - GL_STREAM_DRAW, GL_STREAM_READ, GL_STREAM_COPY, GL_STATIC_DRAW, GL_STATIC_READ, GL_STATIC_COPY, GL_DYNAMIC_DRAW, GL_DYNAMIC_READ, GL_DYNAMIC_COPY
    - Namapovat existující data v prostoru CPU – pro časté změny
      - změny jsou automaticky přenášeny do GPU, před použitím nutné glUnmapNamedBuffer(…)

```
void * glMapNamedBuffer(GLuint buffer, GLenum access)
```

      - GL_READ_ONLY, GL_WRITE_ONLY, GL_READ_WRITE
      - glMapNamedBuffer(…) + {change_data} + glUnmapBuffer(…)

Výběr colorbufferu pro kreslení

- `glDrawBuffer(GLenum buffers)`
  - až 4 najednou
  - použije aktuální framebuffer
- glNamedFramebufferDrawBuffer(GLuint framebuffer,GLenum buf)
  - libovolný buffer z libovolného framebufferu
- Některé buffery
  - double-buffering
    - GL_FRONT, GL_BACK
  - stereoskopické vykreslování
    - GL_FRONT_RIGHT, GL_BACK_LEFT, …
  - neobvyklé: oba najednou (pozadí, neměnné části...)
    - GL_FRONT_AND_BACK
- Zpětné čtení (např. screenshot)

```
glReadPixels(x_start, y_start, width, height, format, type, *pixels)
```

Zjišťování vlastností bufferu

- `glGetIntegerv(GLenum pname, GLint  *vysledek )`
  - zjistí jeden parametr
    - GL_RED_BITS, GL_GREEN_BITS, GL_BLUE_BITS, GL_ALPHA_BITS, GL_DEPTH_BITS, GL_ACCUM_RED_BITS, …
- `int glfwGetWindowAttrib(GLFWwindow *window, int attrib)`
  - součást knihovny GLFW, jiná množina parametrů

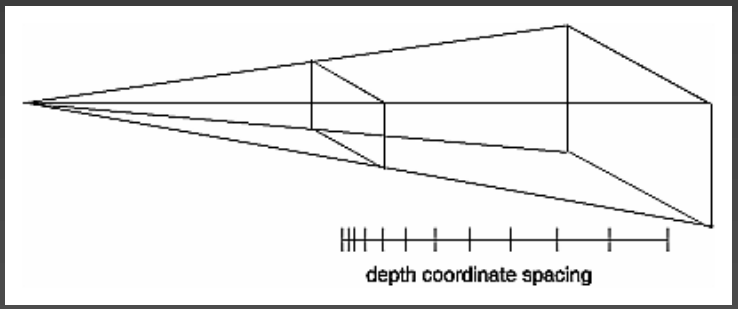
    GLFW_FOCUSED, GLFW_MAXIMIZED, ...

Color buffer

- Nejméně jeden
- Barevná informace fragmentů
  - použité pro vykreslování
  - většinou RGBA (paleta je zastaralá)
- Pro správné vykreslování dva buffery (ev. více)
  - double-buffering: GLFW_DOUBLEBUFFER
- Pro stereoskopické vidění dva (čtyři) buffery
  - levý a pravý: GLFW_STEREO, GL_LEFT, GL_RIGHT
- Možné i další – omezeno jen kapacitou paměti

Depth buffer

- Paměť hloubky, většinou jen jedna
- Rozsah near…far mapován na 0.0f … 1.0f
  - standardně mazáno hodnotou 1.0 (= far plane)
    - přenastavit lze: glClearDepth(GLdouble depth)
- Pro vykreslování viditelných částí těles
  - zakrytá část má větší hloubku a nevykreslí se
- Lze různě nastavovat a používat pro efekty
  - stíny (nejdou přímo) apod.
- Důležité pohlídat si bitovou hloubku
  - může být i jen 8 bitů = 256 úrovní
    - artefakty
  - dnes spíše 16b, 24b, 32b, …

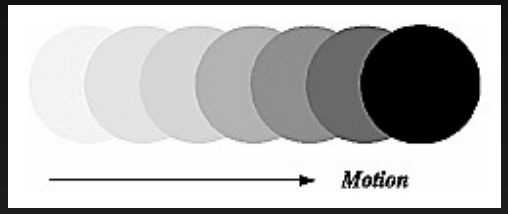


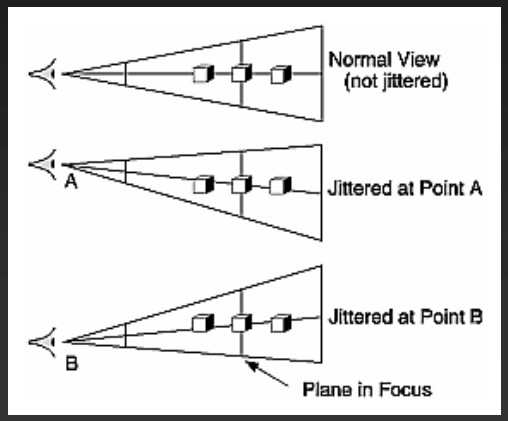
Stencil buffer

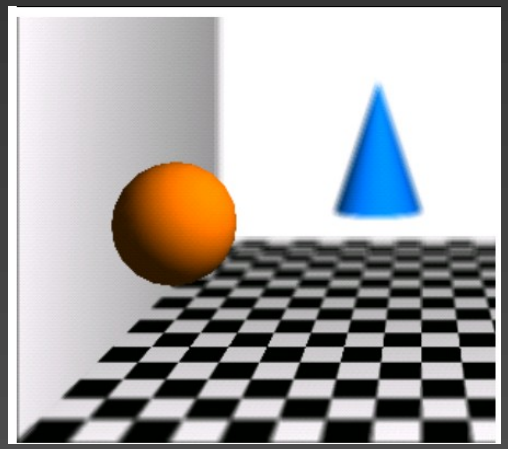
- Šablona pro maskování fragmentů
  - vylučuje zápis zamaskovaných fragmentů
- Rozdíl proti ořezu
  - ořezávání = vrcholy
  - šablona = fragmenty
- Funkce pro maskování je měnitelná
  - CSG, stíny, GUI...
- Většinou stačí 1b hloubka (maska), nebo 8b (rychlost)

Možné využití bufferů

- FBO + zpracování → colorbuffer
- Sloučení více obrazů do jednoho
  - Pro efekty – motion blur, depth-of-field (DOF), apod.
  - Antialiasing – do FBO vykreslení s vyšším rozlišením, pak průměrování
- Scéna vytvořena v samostatném FBO
  - přesun + další zpracování…
  - … do colorbufferu…
  - … swap buffers a zobrazení





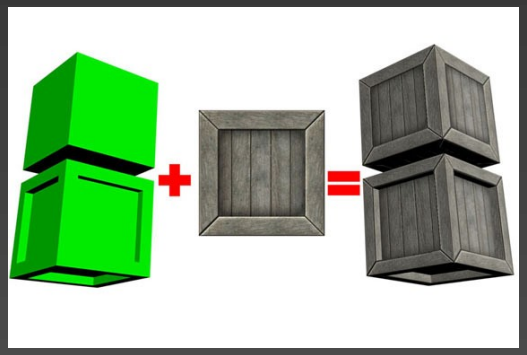


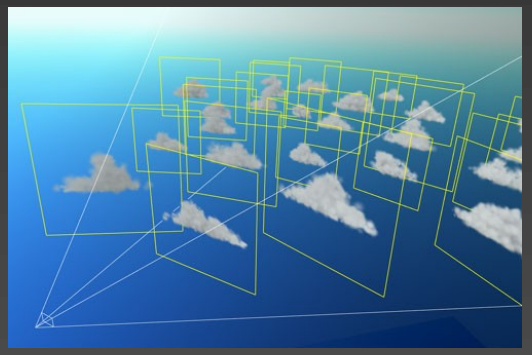
Textury

Textury

- Na danou geometrii nanášíme obrázek
  - v podstatě použije texturu jako zdroj per-fragment difuzního materiálu

- Pro vykreslení složitých těles bez nutnosti modelování přes plošky
  - tráva, stromy, mraky, kameny, zdi,...
  - nejsou to plošky – ztrácíme prostorovou informaci
  - billboarding – správně orientovaná průhledná textura





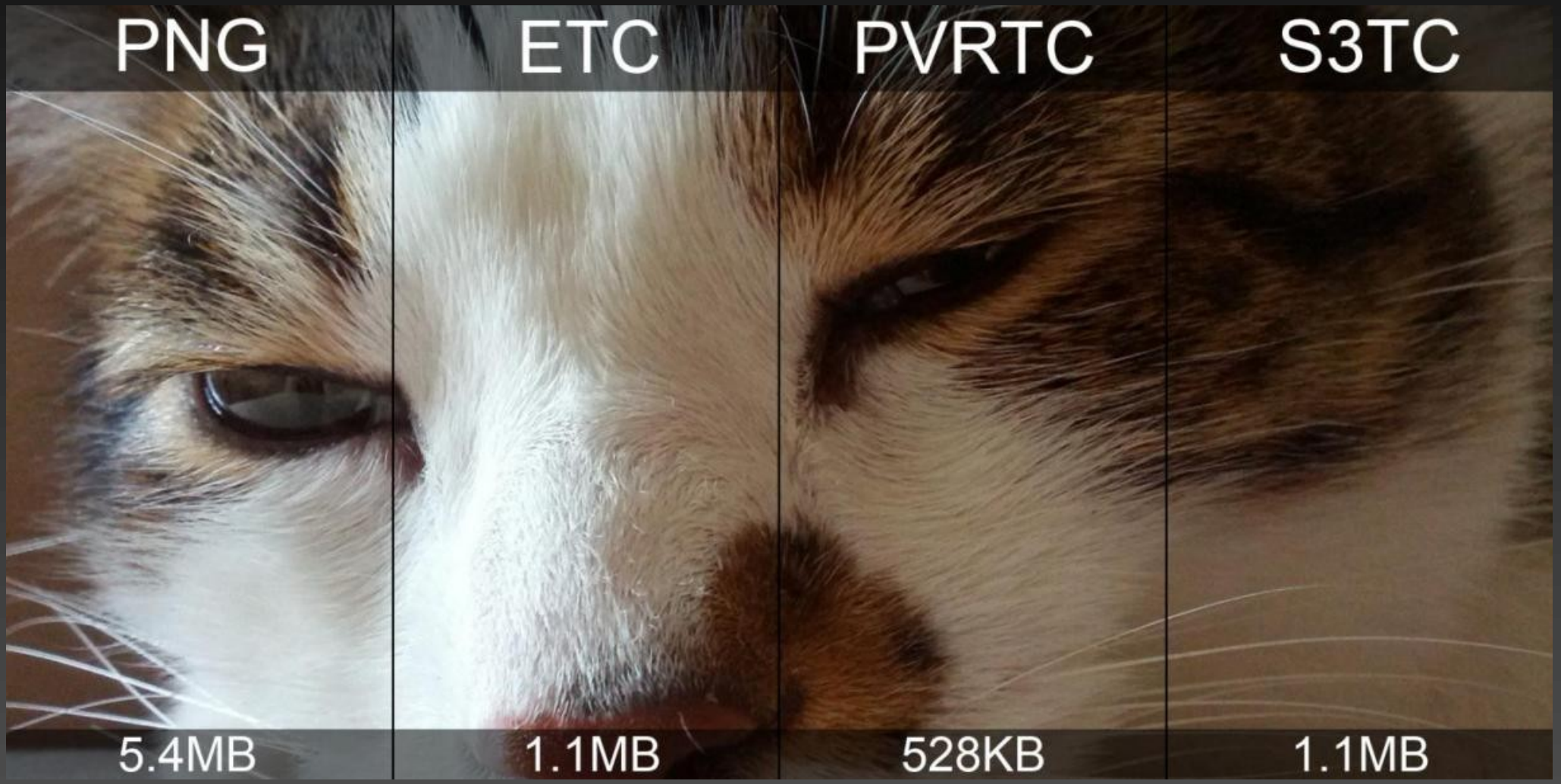
Speciální rastrové formáty – textury

Potřebujeme náhodný, malý kousek velkého obrázku (textury). Nemáme dost paměti ani času na kompletní dekompresi → PNG, JPEG atd. nelze použít!

- Potřebujeme
  - náhodný přístup, rychlost, dobrý kompresní poměr, kvalitu
- Obvykle dlaždicový ztrátový formát. Každá malá dlaždice nezávisle komprimovaná s podporou HW → takřka náhodný přístup, rychlé, efektivní.

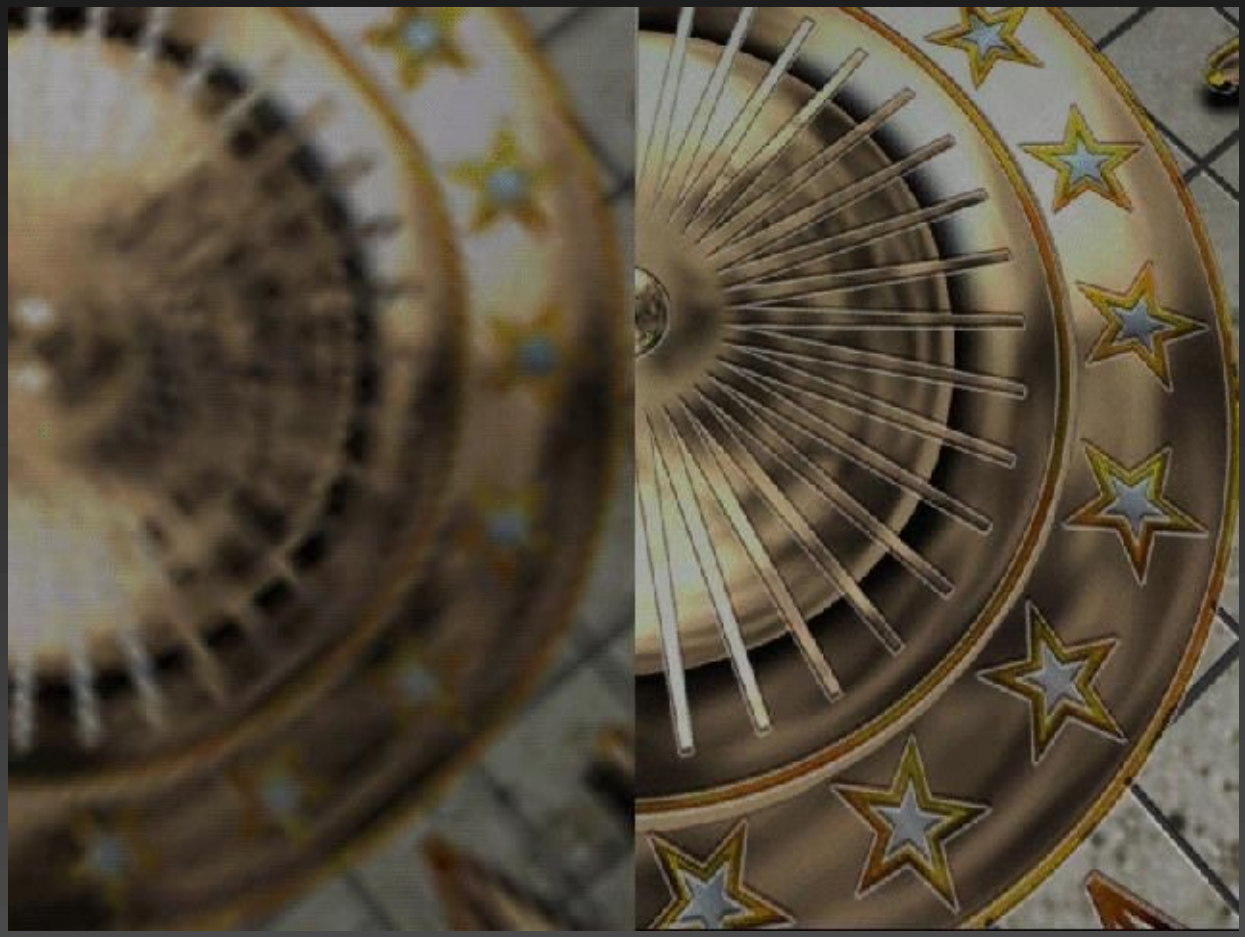
- S3TC, DDS, DXT, PWR, ETC, PVRTC, ASTC, BC, ATC, ...

- ETC – Ericsson Texture Compression (Android)
- PVRTC – PowerVR Texture Compression (Intel, Kindle, …)
- ATITC – ATI Texture Compression (Qualcom, Nexus)
- a další (3DC, Arm ASTC, …)



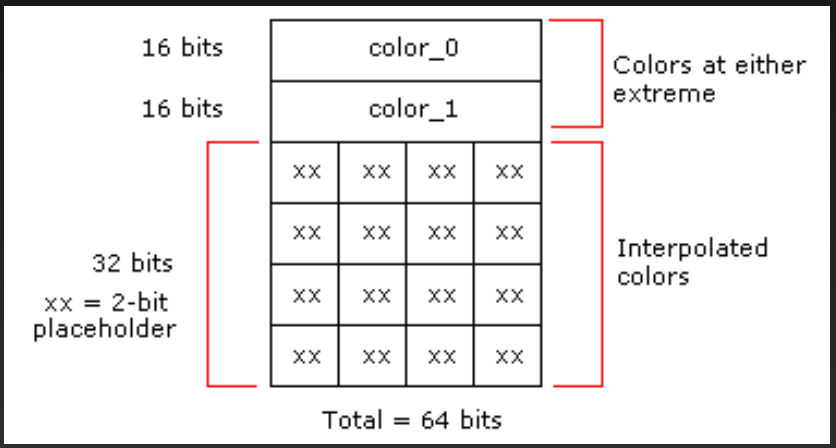
Srovnání kvality

- Nekomprimovaný obraz vs. S3TC stejné velikosti v byte.
- Nekomprimovaný je nutné uložit ve velmi malém rozlišení a při vykreslování zvětšit.



S3TC

- Nejběžnější, jednoduchý (ale překonaný); 5 variant (DXT1-5)
- Bloky 4x4 pixelů R8G8B8 komprimuje na 64 bitů
  - nekomprimovaně 4x4x24 = 384 bitů → komprese 1:6



color_0, color_1 = R5G6B5 (vybraná libovolně)

- if C0 > C1 then C2 = 2/3 * C0 + 1/3 * C1
  C3 = 1/3 * C0 + 2/3 * C1

  else
  C2 = 1/2 * C0 + 1/2 * C1
  C3 = black

Vlastnosti textur

- Typy textur
  - 1D, 2D (nejčastější), 3D
  - rastrové – běžný obrázek
  - procedurální – vzniklé matematickým výpočtem
    - výpočet předem
    - výpočet za běhu (shadery, volumetrické efekty, nižší rozlišení)
- Výhody
  - dostatečně kvalitní pro oklamání oka
    - zeď, kámen, písek, dřevo...
  - jednoduchá hardwarová implementace
- Nevýhody
  - rozlišení: nutné zvolit správný poměr kvalita / obsazená paměť
  - Rastr → filtrace + antialiasing → zpomaluje (přídavné paměťové přenosy)

Postup při texturování

1. Načtení nebo výpočet textury
2. Vytvoření texturovacího objektu
   - Nastavení formátu a parametrů
3. Propojení textury a texturovací jednotky
   - Odeslání uniform s číslem aktivní tex. jednotky
4. (kód shaderu) Výběr způsobu nanášení
5. Vykresli vertexy s novým atributem
   - texturovací souřadnice

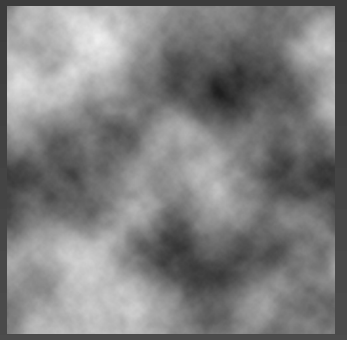
(1/5) Načtení (výpočet) textury

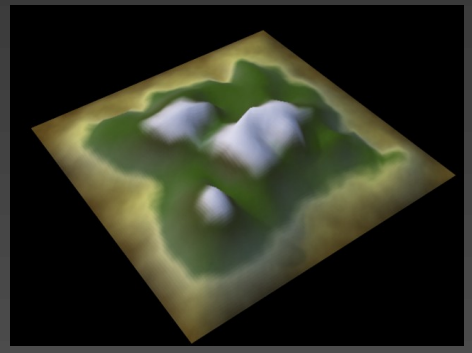
- Není přímá podpora pro textury → knihovny
  - my máme OpenCV 👍
- Musí být: čtverec, rozměr 2^n

NPOT: `glewIsSupported(„GL_ARB_texture_non_power_of_two“);`
`glewIsSupported(„GL_ARB_texture_rectangle“);`
(texturovací souřadnice nejsou <0..1>x<0..1> ale <0..w>x<0..h> a další omezení…)

- Procedurální textury
  - předepsány rovnicemi, výpočet obvykle náročnější než načtení
  - vysoká kvalita (libovolné rozlišení)
  - volumetrické efekty: kouř, oheň
  - fraktály (Perlinův šum apod.): krajina, mraky...

```
#include <glm/gtc/noise.hpp>
glm::simplex(…)
glm::perlin(…)
```





(2/5) Vytvoření texturovacího objektu

- Vytvořit a inicializovat objekt textury, nastavit vlastnosti

```
GLuint texName;

glCreateTextures(GL_TEXTURE_2D, 1, &texName)
```

- parametry aplikace textury (viz dále)

```
glTextureParameteri(texName, … )
```

- nahrání vlastních dat textury
  - vytvoření prázdné oblasti dat (immutable format)

```
void glTextureStorage2D(GLuint texture, GLsizei level,
                        GLenum internalformat, GLsizei width,
                        GLsizei height)
```

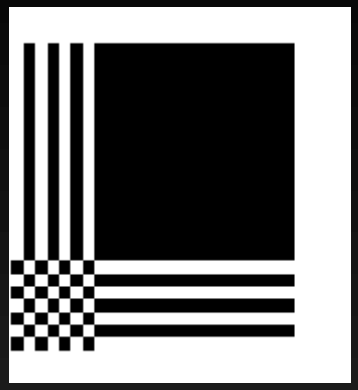
level = mipmap level; format = uvnitř GPU, většinou GL_RGBA8

  - nahrání vlastních dat (mohou být později měněna)

```
void glTextureSubImage2D(GLuint texture, GLint level,
                         Glint xoffset, GLint yoffset,
                         GLsizei width, GLsizei height,
                GLenum format, GLenum type, const void *pixels)
```

format = data C++, většinou GL_BGR(A); type = většinou GL_UNSIGNED_BYTE

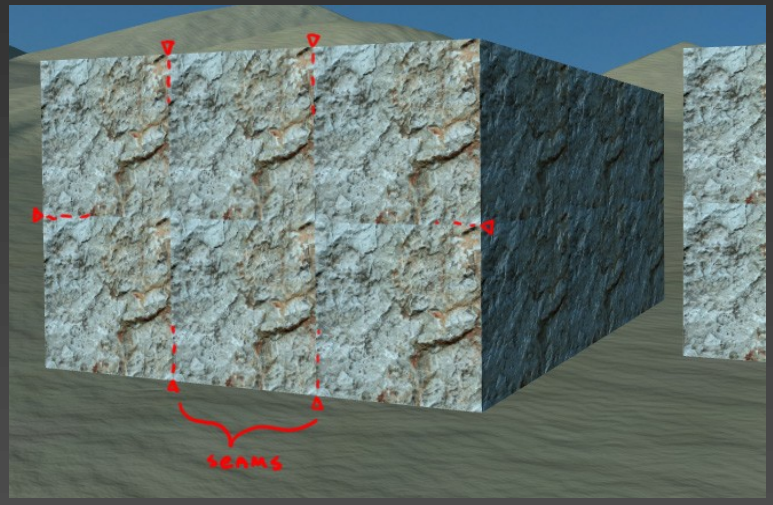
Parametry nanášení



- glTextureParameter{if}{v}(texName, param, val)

- Opakování (GL_TEXTURE_WRAP_S, _T,_R)
  - jako dlaždice (GL_REPEAT, GL_MIRRORED_REPEAT)
  - opakovat krajní hodnotu (GL_CLAMP_TO_EDGE)
  - vyplnit zbylou oblast barvou (GL_CLAMP_TO_BORDER)
    - additional texture border color GL_TEXTURE_BORDER_COLOR

- Filtration (GL_TEXTURE_MIN_FILTER)
  - žádná = nejbližší soused (GL_NEAREST)
  - bilinear (GL_LINEAR)
  - trilinear (GL_LINEAR_MIPMAP_LINEAR)

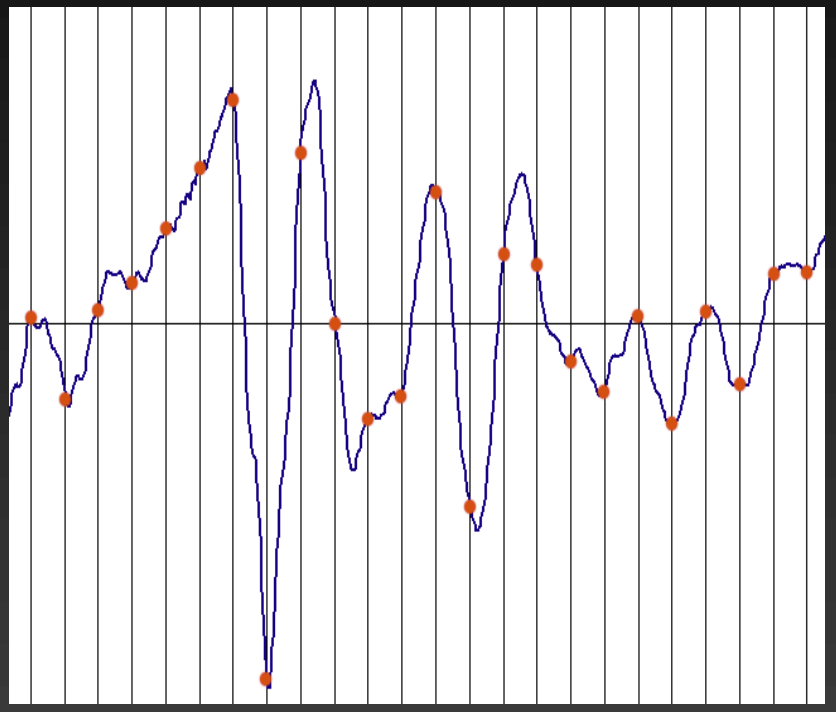


Filtrace - resampling - převzorkování

- Vzorkovat lze jen spojitý signál...
  - interpolace: Diracův impuls jako digitalizační filtr
  - filtrace: digitalizační filtr s nenulovou plochou (např. čtverec, konus)
- … ale data jsou už navzorkovaná (rastr).
  - hledáme spojitou aproximaci diskrétního signálu
  - aproximace může být znovu navzorkována s jinou frekvencí

Rekonstrukce

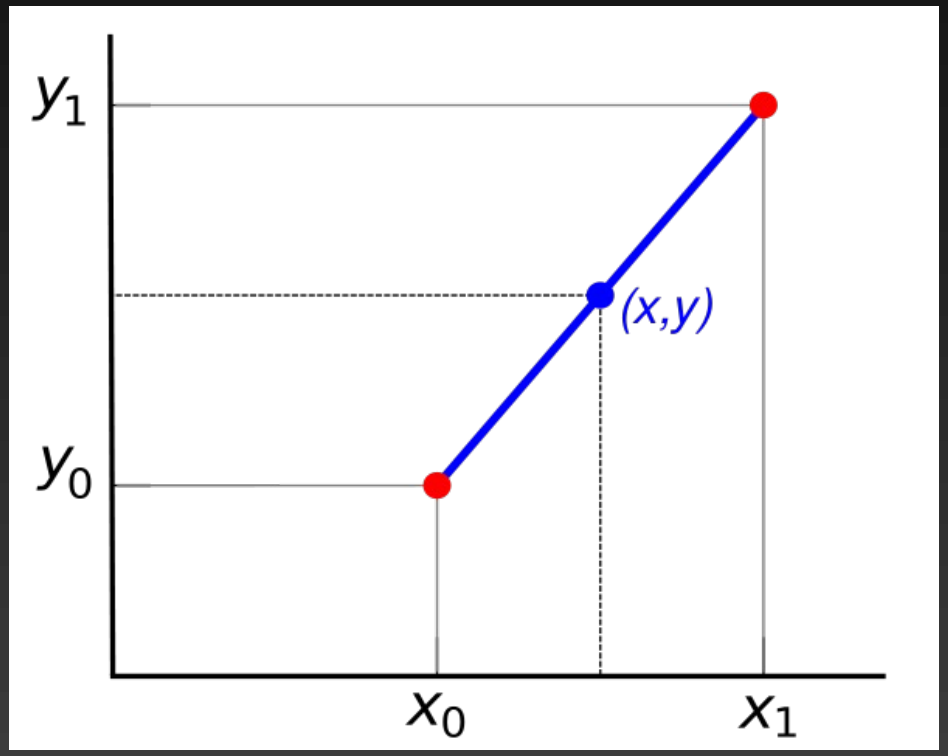
- Získání spojitého signálu z už navzorkovaného
  - Nearest neighbour
  - Linear (bilinear)
  - Cubic (bicubic)
  - Lanczos
  - Sinc (optimum, teor.)



Lineární interpolace

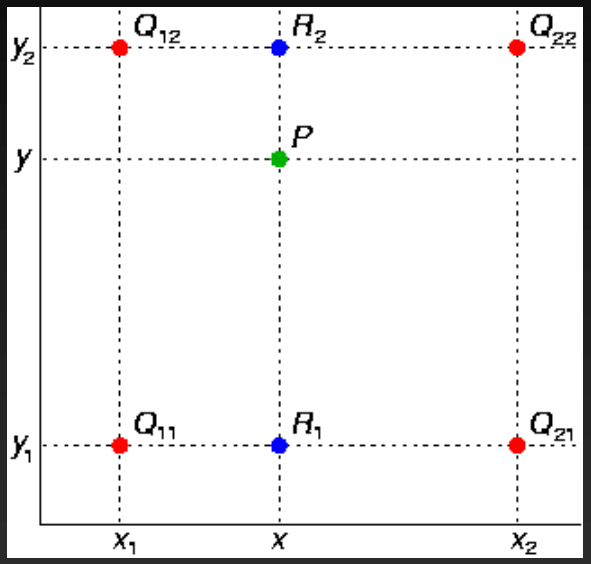
- Interpolovaná hodnota vybraná proporcionálně podle vzdálenosti od původních hodnot
- Nejjednodušší

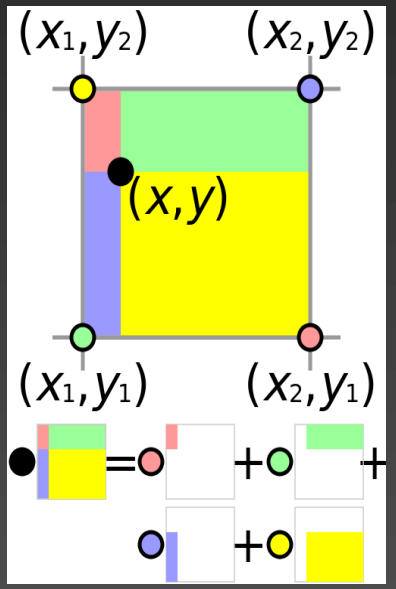
(př. pro jednu dimenzi)



Bilineární interpolace

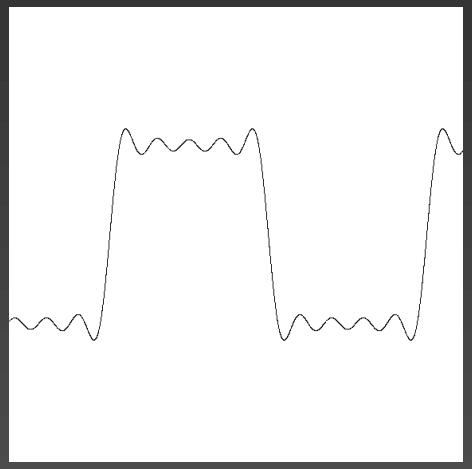
- Lineární interpolace dvou proměnných
  - X, Y → 2D
- 4 sousední vzorky
- 3 lineární interpolace
- Rozmazává hrany (text, čáry, …)

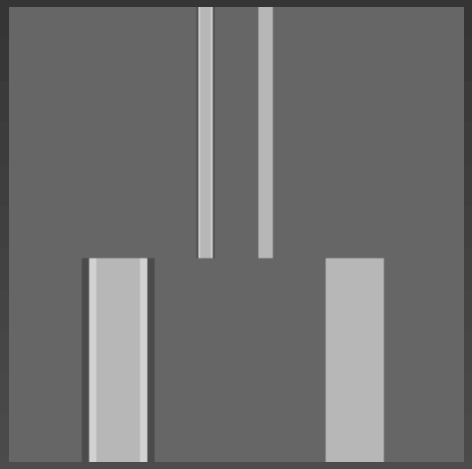


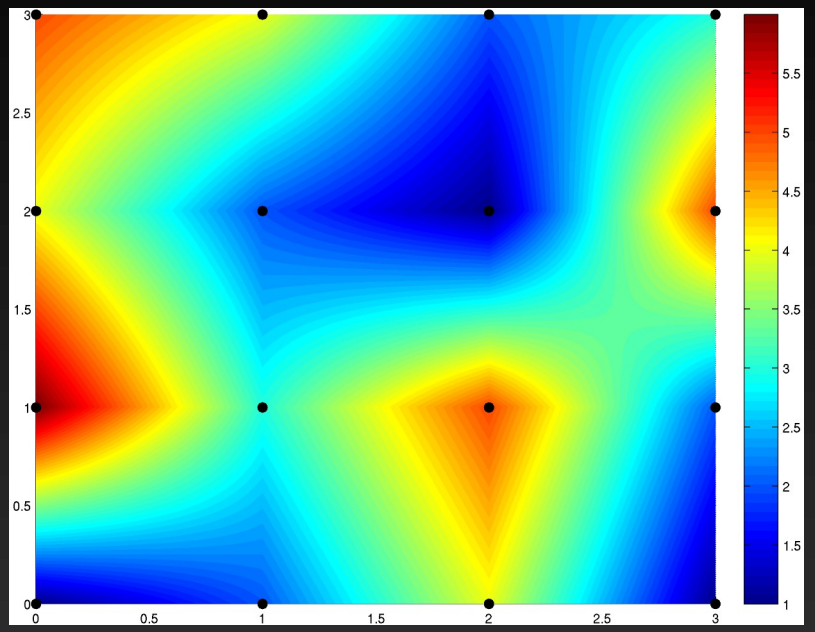


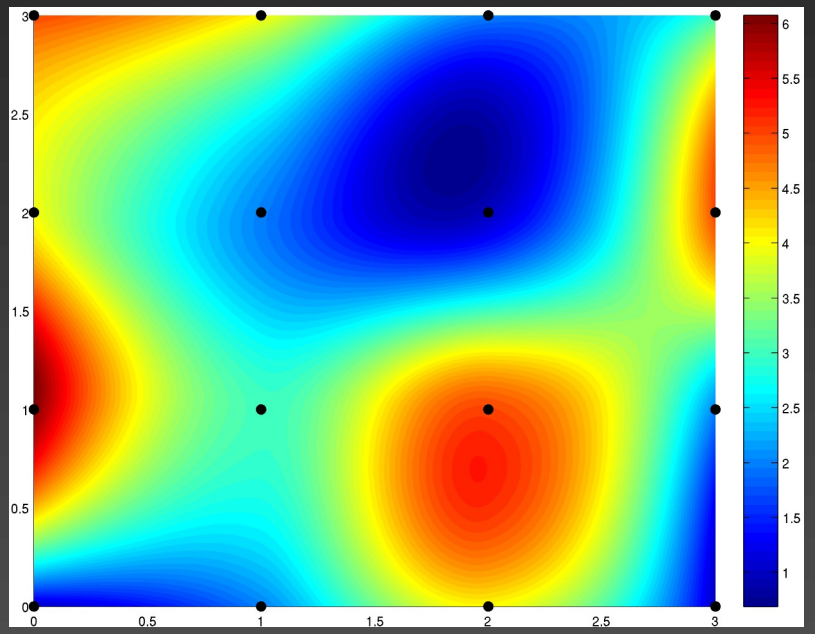
Bikubická interpolace

- Spline-based
- 16 sousedních vzorků samples
- výpočetně náročná
  - 16 rovnic o 16 neznámých
- Plynulejší
- Méně artefaktů
  - zachovává detaily
- Generuje překmity





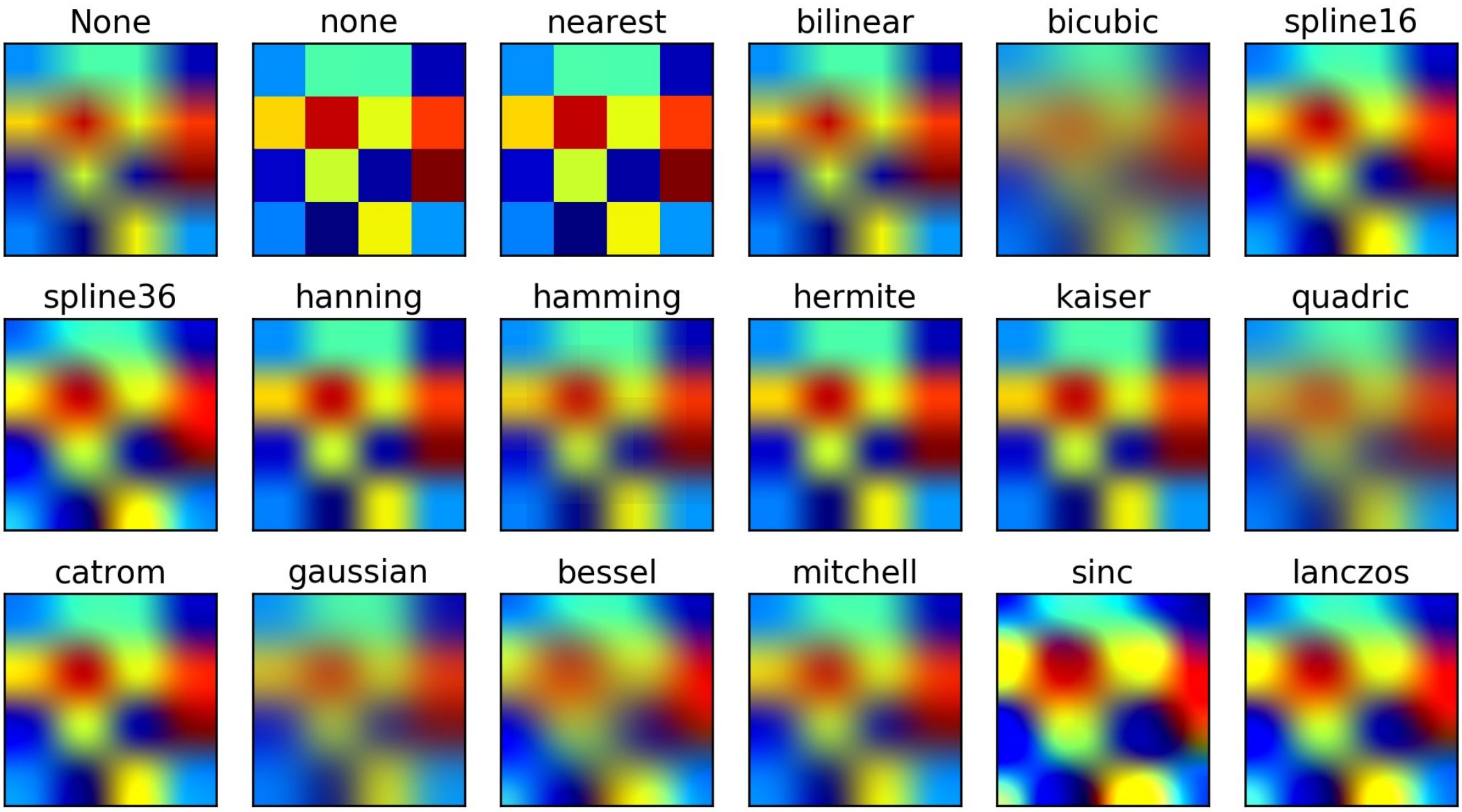




Změna rozlišení

- Rekonstukce obrazu převzorkováním
  - nejbližší soused (nearest neighbour)
  - bilineární interpolace
  - kubická interpolace, atd.
- Nová vzorkovací frekvence může být
  - vyšší → zvětšení
  - nižší → zmenšení
- Interpolátory neustále využívány v GPU
  - specializovaný HW blok

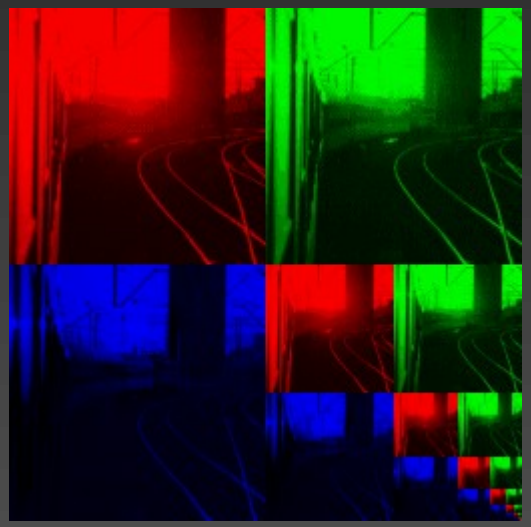
Různé metody interpolace

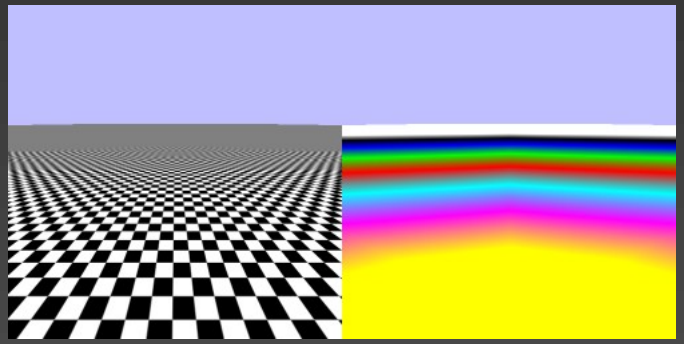


Mipmapping

- Pro snížení (odstranění) aliasu
- Pouze pro zmenšování
- Lineární interpolace ve třech osách
  - v textuře – s, t
  - mezi MIPMAP texturami podle Z vzdálenosti

glGenerateTextureMipmap(texName)



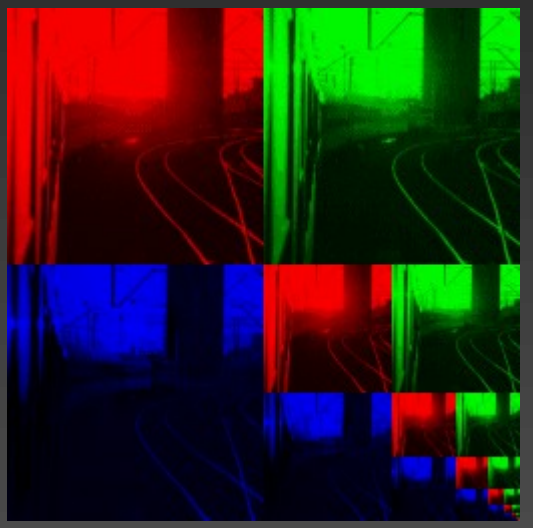


MIP map

- „multum in parvo“ - mnoho v malém
- Hierarcická reprezentace obrázku
  - spočítaná a filtrované předem, ne v reálném čase

+ Při zmenšení zvyšuje kvalitu
  - šetří čas → zrychluje vykreslení (připravena offline, menší)
  - vyšší kvalita (lepší filtrace)
  - mapování textur na vzdálené objekty

- Spotřeba paměti
  - 4/3 původní spotřeby
  - uložena v jediném obrázku
    - menší fragmentace paměti
  - nutné správně vytvořit, volit souřadnice



Další možnosti

- Anisotropní filtrace
  - zlepšuje kvalitu šikmých povrchů
    - více vzorků v jednom směru
  - výpočetně náročná

Core in version	4.6
Core since version	4.6
Core ARB extension	ARB_texture_filter_anisotropic
EXT extension	EXT_texture_filter_anisotropic

```
GLfloat maxAniso = 0.0f;
glGetFloatv(GL_MAX_TEXTURE_MAX_ANISOTROPY_EXT, &maxAniso);

glSamplerParameteri(sampler, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glSamplerParameteri(sampler, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glSamplerParameterf(sampler, GL_TEXTURE_MAX_ANISOTROPY_EXT, maxAniso);
```

(3/3) Propojení texturovací jednotky

- aktivovat shader
  ```
  glUseProgram(shaderProgram)
  ```
- propojit (bind) texturu a texturovací jednotku (např. č. 0)
  ```
  glBindTextureUnit(0, texName)
  ```
- odeslat vybranou texturovací jednotku do shaderu

  ```
  glProgramUniform1i(shaderProgram,
      glGetUniformLocation(shaderProgram, "tex"), 0);
  ```
- počet texturovacích jednotek je omezený
  - limituje počet souběžně aktivních textur, při překročení nutný rebind

    glGetIntegerv(GL_MAX_TEXTURE_UNITS, &num)

(4/5) Způsoby nanesení textury (kód shaderu)

- Vytvořit sampler = specializovaná proměnná, reprezentuje HW blok texturovací jednotky

```
uniform sampler2D myTexUnit; // texture unit from C++
```

- Vybrat způsob nanesení
  - Modulace
    - nejobvyklejší, barva podkladu (materiál) násobená texturou
      $RGBA = RGBA_{material} * RGBA_{texture}$
    - `vec4 color_out = diffuse_color * texture(myTexUnit,coords)`
    - barva materiálu obvykle není zadávána konstantou, ale pochází z osvětlovacího modelu s ambient, diffuse a specular komponentou (viz další přednášky)

```
vec4 color_out = (light_ambient + light_diffuse) * texture(myTexUnit,coords)
                 + light_specular * specular_material;
```

  - Přímá aplikace
    - data textury přímo nanesena, včetně průhlednosti

```
vec4 color_out = texture(myTexUnit, coords);
```

(5/5) Vykreslení s použitím texturovacích souřadnic

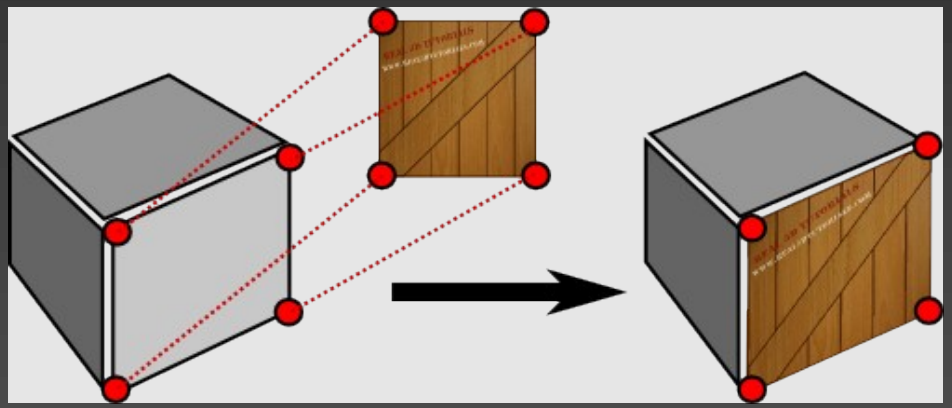
- Vykreslit – VAO musí obsahovat texturovací souřadnice

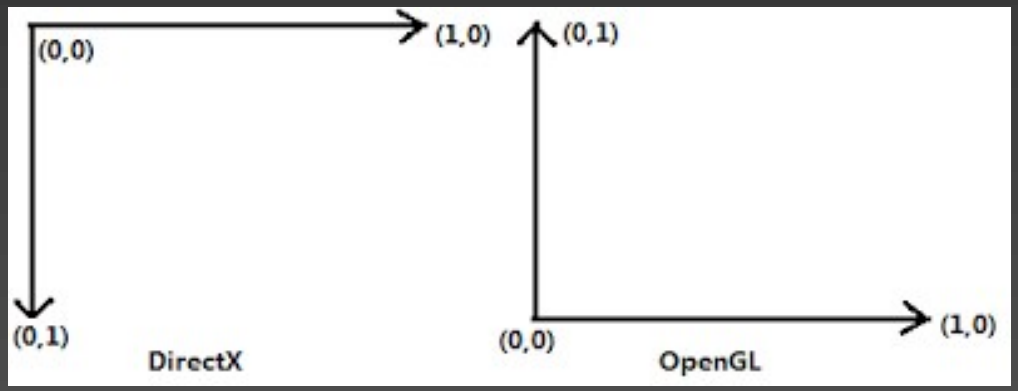
```
glBindVertexArray(VAO);
glDrawElements(GL_TRIANGLES, indices.size(), GL_UNSIGNED_INT, 0);
```

- Normalizované texturovací souřadnice
  - textura je definována jako čtverec o straně 1.0
  - obecně (s, t, r, q)
    - nejčastěji 2D textury → jen (s, t), automatické doplnění (t=0.0, q=1.0)
    - rozpory v terminologii: ST vs. UV souřadnice (UV coordinates)

- Multitexturing
  - aplikace více textur najednou

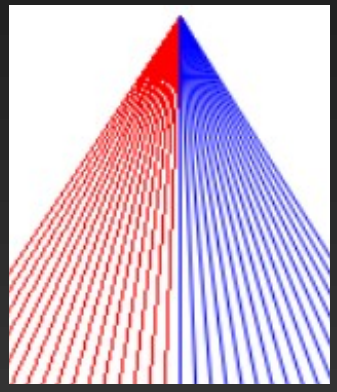
```
vec4 out_color = texture(texunit1,texcoord1) * texture(texunit2, texcoord2)
```

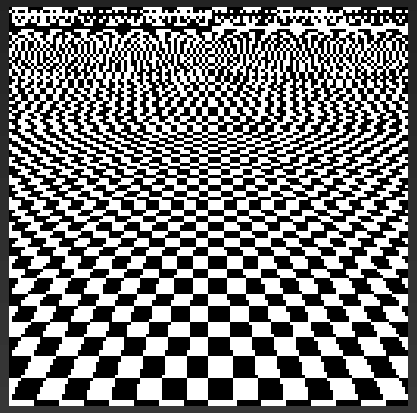


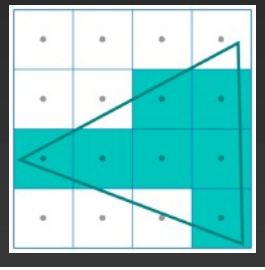


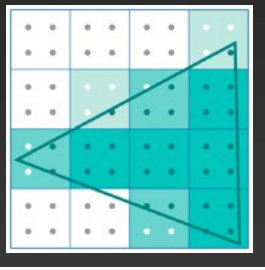
Antialiasing

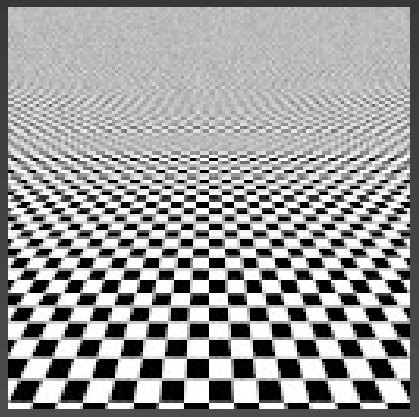
- Malé rozlišení výstupního zařízení způsobuje alias
  - zubaté hrany, přetrhané tenké linie
- Vyhlazování hran pomocí průměrování více vzorků – samplů
  - sample = vzorek z testovacího bodu, který pro fragment rozhodne mezi barvou popředí / pozadí

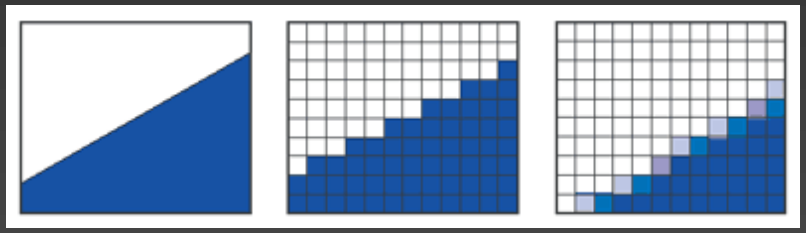










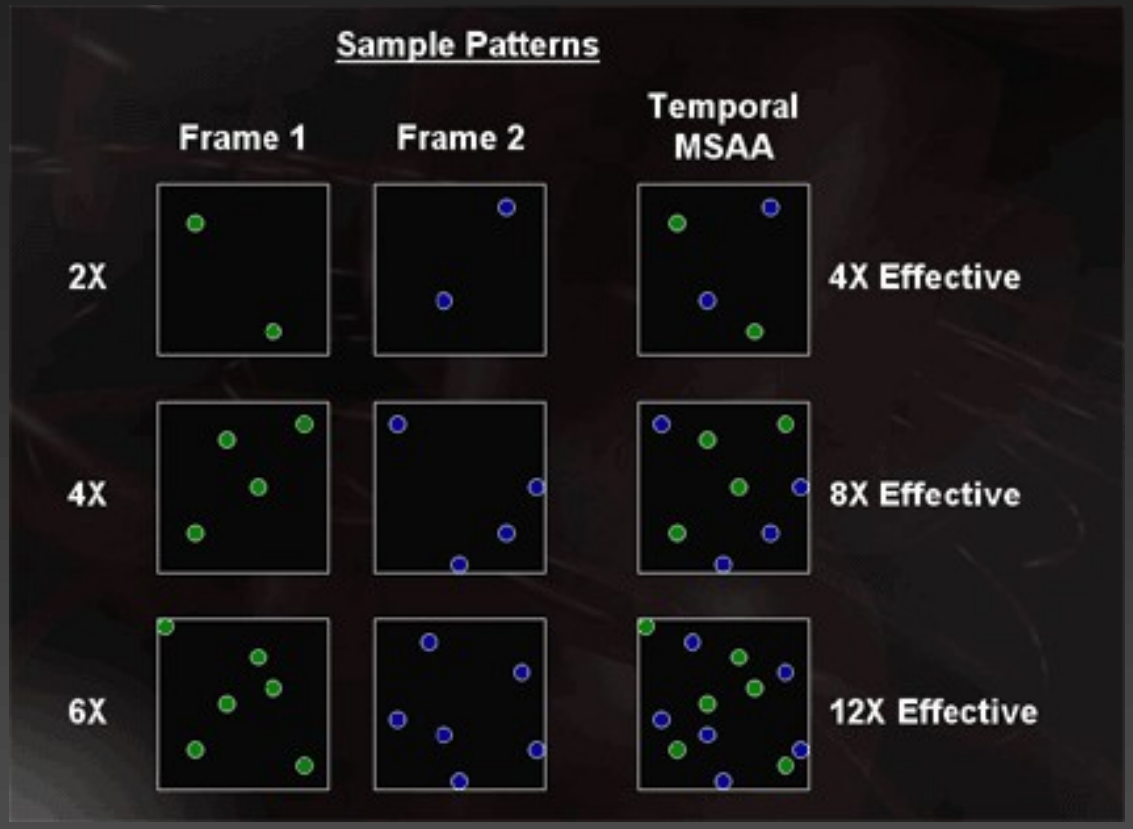


Antialiasing

- Temporální – víc vzorků hledáme v čase
  - využívá vzorky předchozích snímků (Nvidia TXAA apod.)
  - čas na snímek = výpočet aktuálního snímku (cca jako bez AA) + postprocessing (kombinace aktuálního a minulého snímku)
- Spatiální – víc vzorků hledáme v aktuálním snímku
  - více vzorků pro jeden pixel v aktuálním snímku
  - vyšší kvalita
  - čas na snímek = výpočet aktuálního snímku s násobným (2, 4, 8) počtem vzorků

Temporální antialiasing

- Vzorky z předchozího snímku (snímků)
  - + malé výpočetní nároky
  - - rozmazává, chvějící se obraz při malém FPS



Supersampling
Full Scene Anti-Aliasing = FSAA

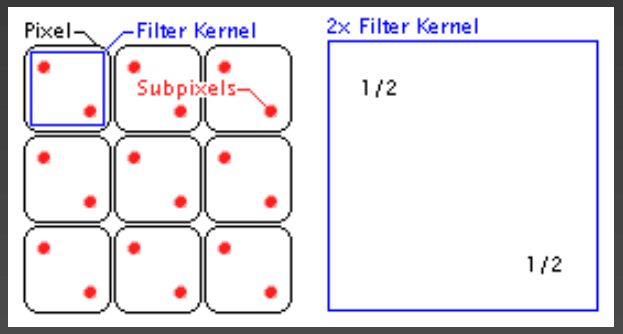
- Maximální kvalita

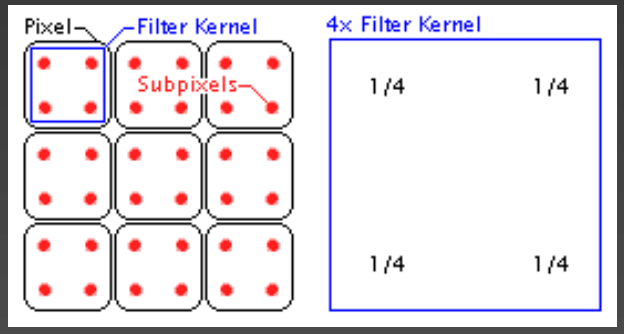
- Standardně je fragment obarven podle vzorku uprostřed pixelu
- 4FSAA – interně 4x více subpixelů na různých místech + avg

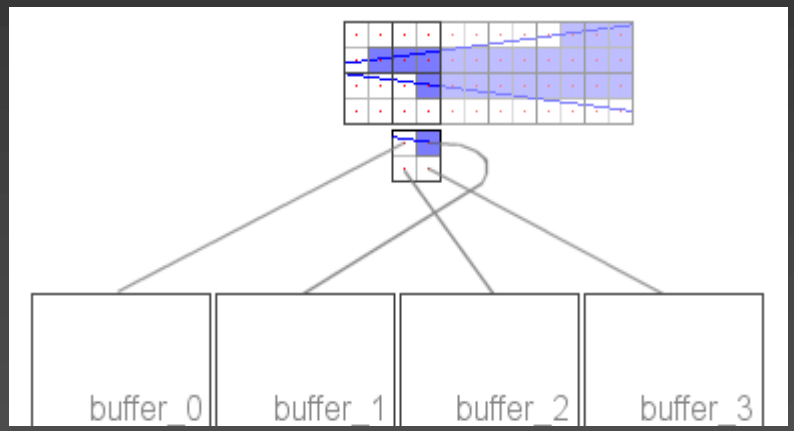
  → fragment shader spuštěn 4x

  \+ vyhlazuje nejen hrany, ale i textury, průhledné textury apod.

  \- značné nároky na paměť (multisample buffers) a výkon







Multisampling

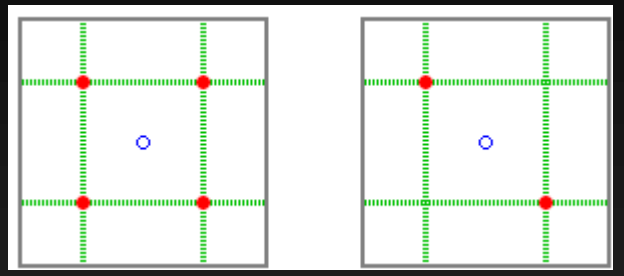
- Optimalizace
  - zjistí se jen příslušnost subpixelu k trojúhelníku
  - následně interpolace vstupů pro fragment shader
    → fragment shader spuštěn pouze jednou
  - rozlišení zvětšené jen pro Z-buffer a stencil buffer
- Mám podporu? GL_ARB_multisample
- Různé metody volby samplů
  - snaha o minimum samplů tak, aby interní rozlišení maximálně narostlo

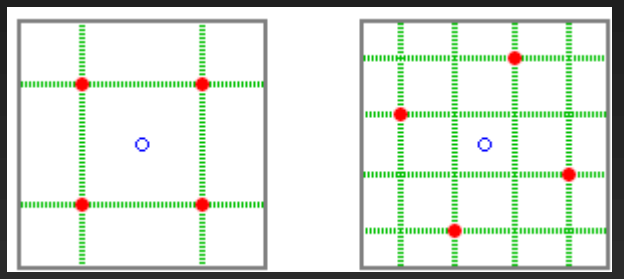
glfwWindowHint(GLFW_SAMPLES, 4)

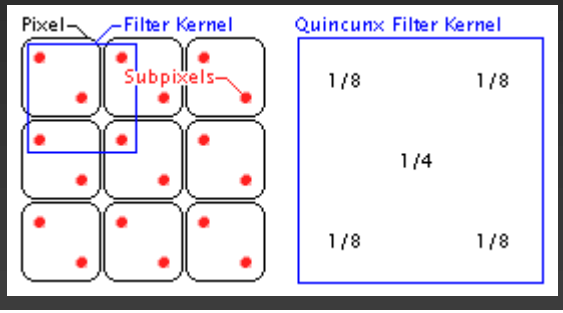
- HINT, t.j. nastavení PŘED vytvořením GL okna!

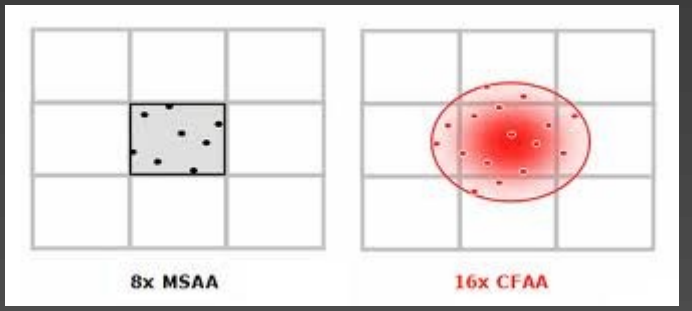
Příklady multisamplingu

- Obdobná kvalita
  (2x nárůst interního rozlišení)
- Lepší volba pozice subsample
  - natočená mřížka = 4x nárůst
- Využití okolních fragmentů
  - mírně rozmazává obraz
- Kombinace postupů



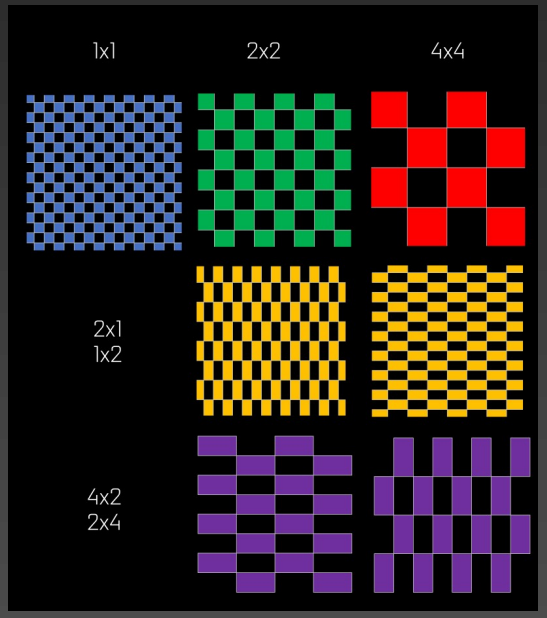


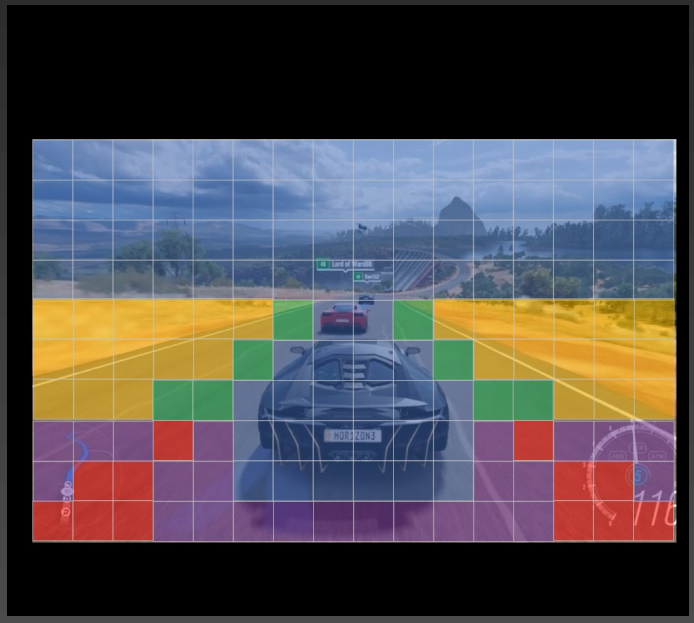


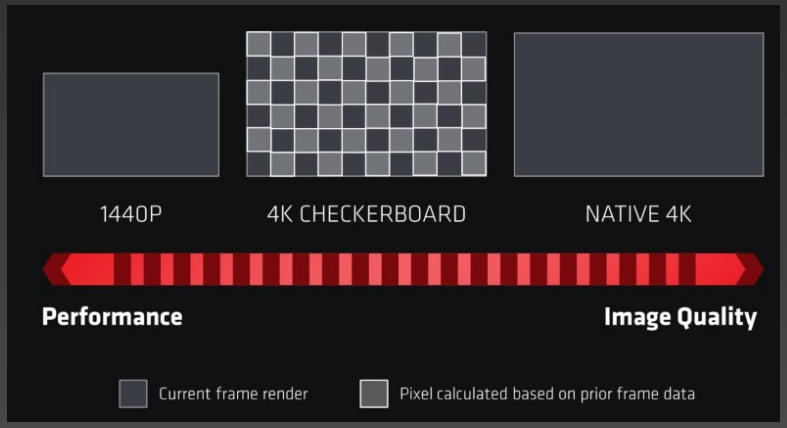


Variable Rate Shading - přeskakování vzorků

- snižuje kvalitu, zrychluje
  - vynecháváme v místech rychlých změn + průměrování s předchozím snímkem
    - snížení kvality uživatel obvykle nepostřehne
  - vynechání lze měnit každý snímek – podle aktuálního FPS

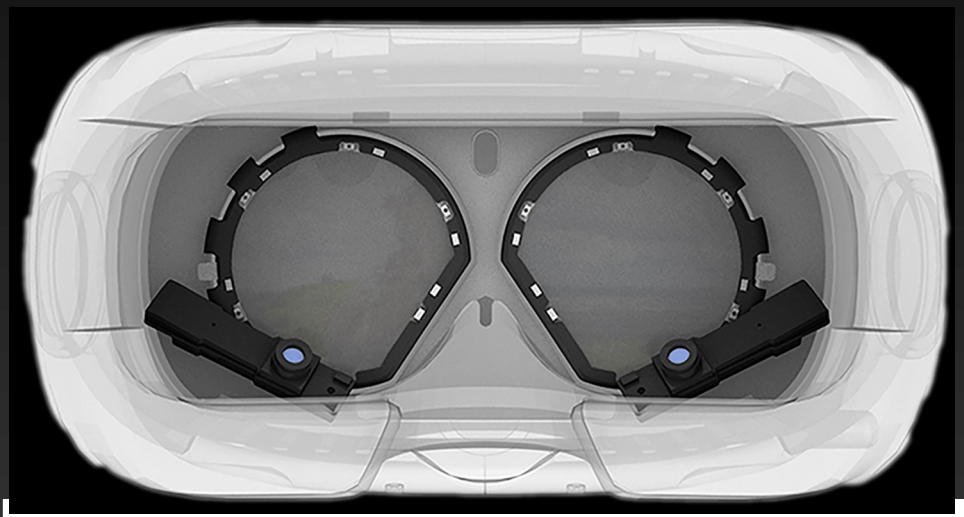






VRS - použití pro VR

- kvalitní obraz jen ve směru pohledu oka
  - nutná rychlá zpětná vazba



Sub-pixel antialiasing

- Náročné na výpočet
- Použití především u písma
  - CoolType, ClearType, FreeType apod.
- Princip – využití R,G,B subpixelů
  - W = (R+G+B)= (B+R+G) = (G+B+R)
  - citlivé na pořadí subpixelů
  - monitor otočený o 90° → problém (win)