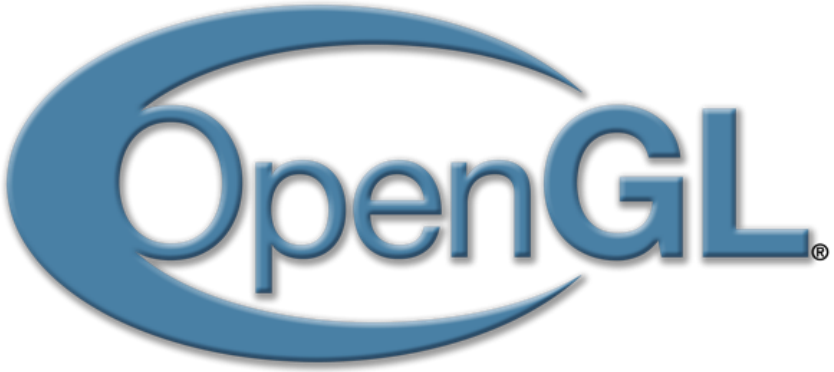


PV112 – Programování grafických aplikací

8. přednáška – Uniform buffer objekt, redukce počtu glDraw*, multisampling

Interface blok

- Seskupuje proměnné dohromady
- Pro *in*, *out*, *uniform*

- Syntaxe:

```
qualifier block_name
{
   members
   ...
   ...
   ...
} [instance_name];
```

Příklad:

```
uniform LightData
{
   vec4 position;
   vec3 ambient;
   vec3 diffuse;
   vec3 specular;
} light;
```

Interface blok

- *instance_name* je nepovinná:

```
out VertexData
{
   vec3 pos_ws;
};

...

pos_ws = M_matrix * pos;
```

```
out VertexData
{
   vec3 pos_ws;
} output;

...

output.pos_ws = M_matrix * pos;
```

in/out interface blok

- Pouze pro proměnné posílané mezi shadery, nikoliv pro vstup do VS nebo výstup z FS

VS:

```
out VertexData
{
   vec3 position;
   vec3 normal;
} outData;
```

FS:

```
in VertexData
{
   vec3 position;
   vec3 normal;
} inData;
```

- *block_name* a proměnné musí být stejné, *instance_name* nemusí

uniform interface blok

- Seskupuje uniformní proměnné dohromady

```
uniform LightData
{
   vec4 position;
   vec3 ambient;
   vec3 diffuse;
   vec3 specular;
} light;
```

- Data proměnných jsou uložené v bufferu (uniform buffer objektu – UBO)

Uniform buffer objekty

- Další typ buffer objektů
- *GL_UNIFORM_BUFFER*
- Vytvoření bufferu, načítání/aktualizace dat jako u jiných bufferů

```
GLuint light_ubo;
glGenBuffers(1, &light_ubo);
glBindBuffer(GL_UNIFORM_BUFFER, light_ubo);
glBufferData(GL_UNIFORM_BUFFER, ...);
...
glBufferSubData(...);
```

Uniform buffer objekty

```
GLuint glGetUniformBlockIndex(
    GLuint program, const char *name)
```

- Získáme index uniform interface bloku
- *program* = shader program
- *name* = jméno **bloku**, nikoliv instance
- Vrací *GL_INVALID_INDEX*, pokud daný blok neexistuje

Uniform buffer objekty

```
uniform LightData { ... };
```

- dotazuji se na *LightData*

```
uniform LightData { ... } light;
```

- dotazuji se na *LightData*, nikoliv na *light*

```
uniform LightData { ... } light[2];
```

- dotazuji se na *LightData*
- získaný index odpovídá *light[0]*
- získaný index +1 odpovídá *light[1]*

Navázání UBO

```
void glUniformBlockBinding(GLuint program,
  GLuint uniformBlockIndex,
  GLuint uniformBlockBinding)
```

- Pro zadaný program sváže index uniform bloku *uniformBlockIndex* se místem *uniformBlockBinding*.
- Můžeme tak navázat uniform buffer vždy ke stejnému místu, ať má blok v daném programu jakýkoliv index

Navázání UBO

```
void glBindBufferBase(GLenum target,
  GLuint index, GLuint buffer)
```

- Navážeme UBO *buffer* na binding point *index*
- *target* = *GL_UNIFORM_BUFFER*

- Zároveň naváže buffer pro operace jako *glBufferData*, *glMapBuffer* apod.

```
LD_loc1 = glGetUniformBlockIndex(program1, “LightData”);
LD_loc2 = glGetUniformBlockIndex(program2, “LightData”);
LD_loc3 = glGetUniformBlockIndex(program3, “LightData”);
glUniformBlockBinding(program1, LD_loc1, 0);
glUniformBlockBinding(program2, LD_loc2, 0);
glUniformBlockBinding(program3, LD_loc3, 0);

...

glBindBufferBase(GL_UNIFORM_BUFFER, 0, light_ubo);

glUseProgram(program1);
glDraw*
glUseProgram(program2);
glDraw*
glUseProgram(program3);
glDraw*
```

Navázání UBO

```
void glBindBufferRange(GLenum target,
  GLuint index, GLuint buffer,
  GLintptr offset, GLsizeiptr size)
```

- Podobná jako *glBindBufferBase*, ale naváže jen část bufferu v rozsahu daném parametry *offset* a *size*

Zarovnání dat v UBO

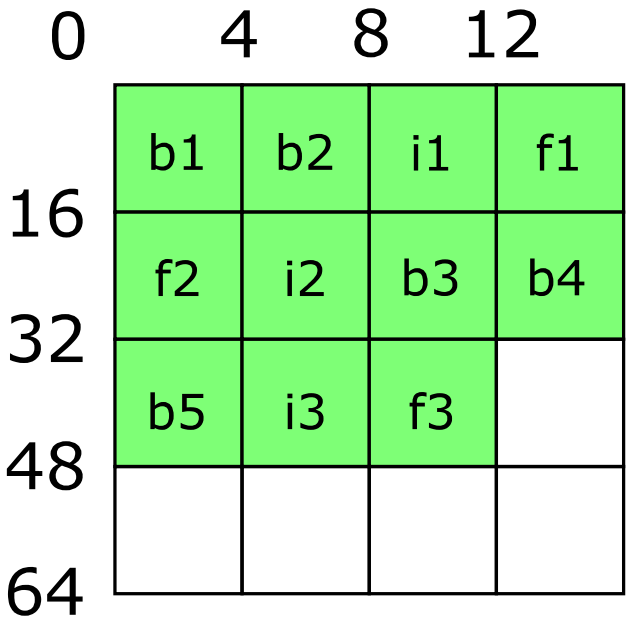
- OpenGL zarovnává jednotlivé proměnné v uniform bloku, aby k nim lépe přistupoval
  - kvůli rychlosti, přístupům do paměti apod.
- Způsob zarovnání se zadává v shaderu pomocí *layout(způsob)*
  - ***std140**, shared, packed, ...*

```
layout (std140) uniform LightData
{
   ...
} light;
```

layout (std140)

- Důležitá je velikost prvků (v bytech) a jejich zarovnání (v bytech)
- *bool*, *int*, *uint*, *float*: velikost 4, zarovnání 4

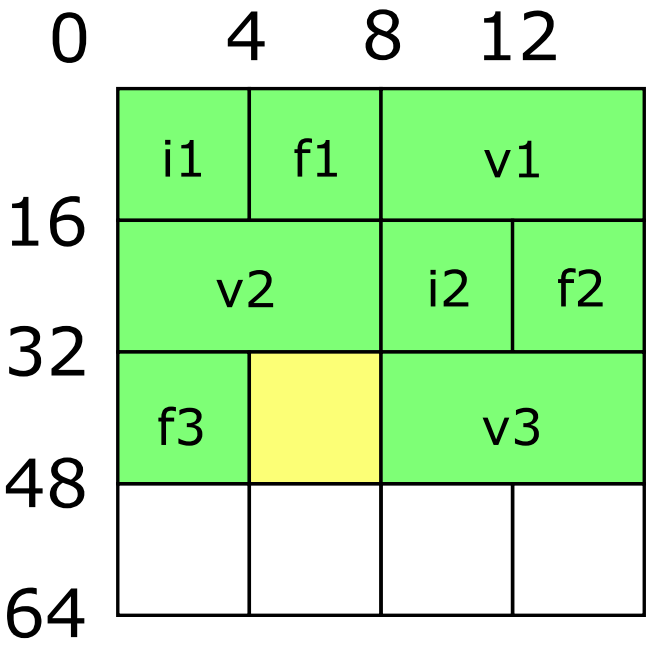
```
layout (std140) uniform U
{
   bool b1, b2;
   int i1;
   float f1, f2;
   int i2;
   bool b3, b4, b5;
   int i3;
   float f3;
};
```



layout (std140)

- **vec2*: velikost 8, zarovnání 8

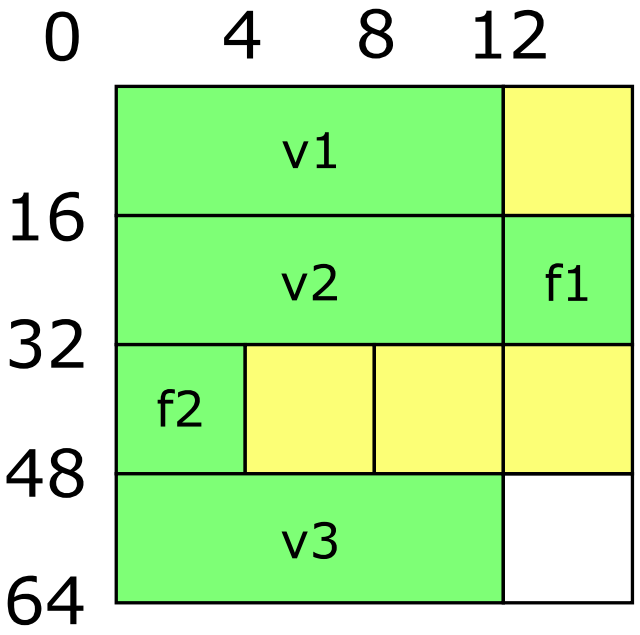
```
layout (std140) uniform U
{
    int i1;
    float f1;
    vec2 v1, v2;
    int i2;
    float f2, f3;
    vec2 v3;
};
```



layout (std140)

- **vec3*: velikost 12, zarovnání 16(!)

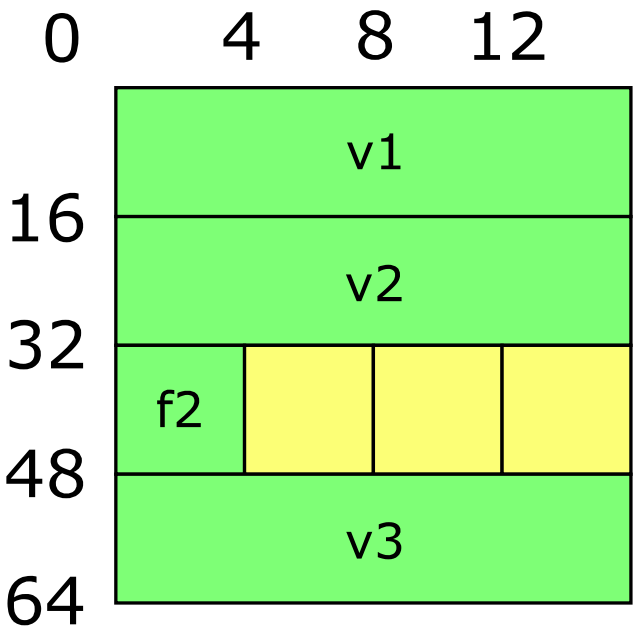
```
layout (std140) uniform U
{
    vec3 v1, v2;
    float f1, f2;
    vec3 v3;
};
```



layout (std140)

- **vec4*: velikost 16, zarovnání 16

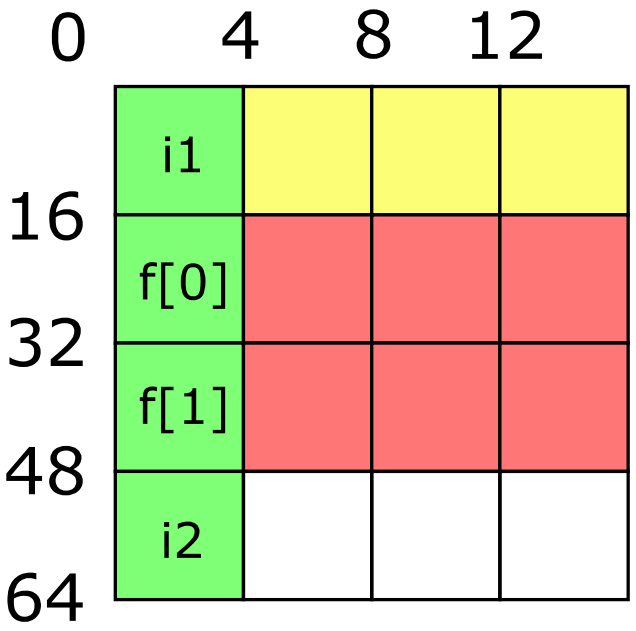
```
layout (std140) uniform U
{
    vec4 v1, v2;
    float f2;
    vec4 v3;
};
```



layout (std140)

- *pole T[n]*: vel. „násobek(16) * n“, zarovnání 16
  - velikost 16*n pro int, float, vec2, …
  - velikost |T|*n pro matice a další

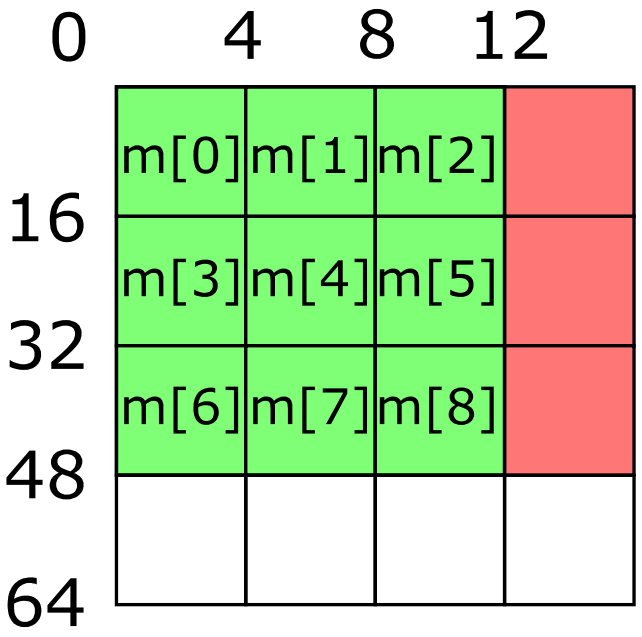
```
layout (std140) uniform U
{
    int i1;
    float f[2];
    int i2;
};
```



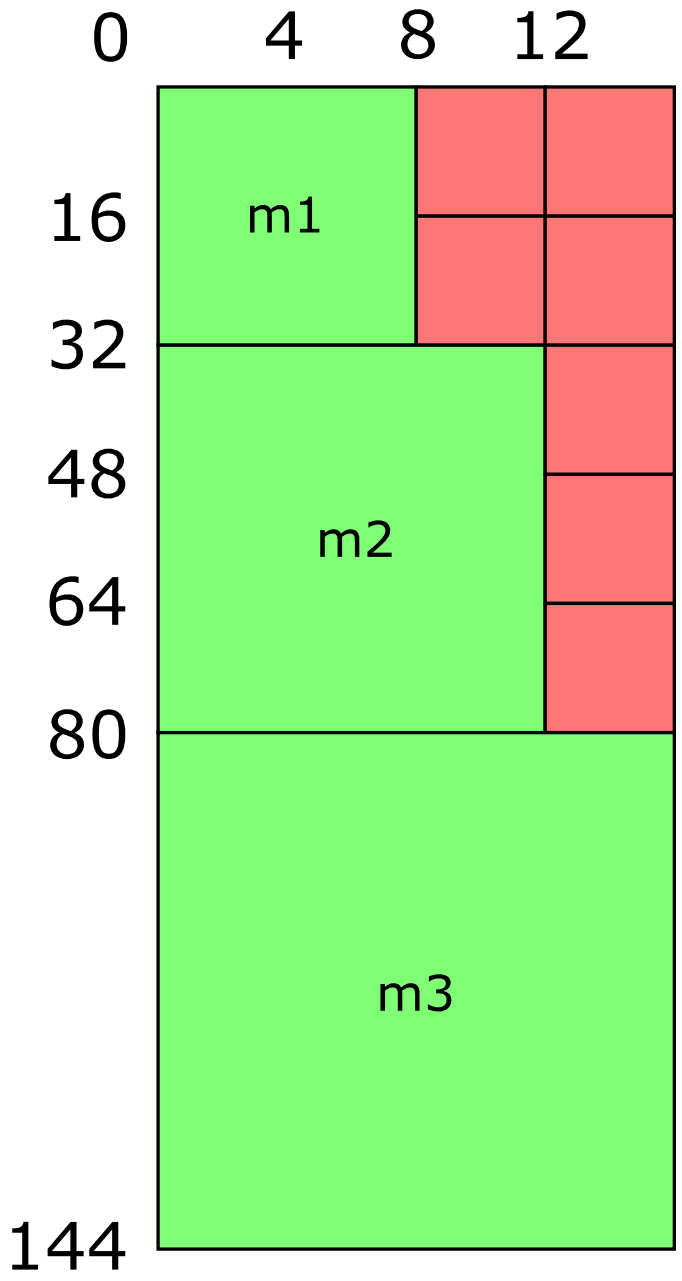
layout (std140)

- **mat*: jako pole vektorů
  - pořadí prvků v matici stejné, jako bychom použili *GL_FALSE* pro *transposed* u *glUniformMatrix*

```
layout (std140) uniform U
{
   mat3 m;
};
```



```
layout (std140) uniform U
{
    mat2 m1;
    mat3 m2;
    mat4 m3;
};
```

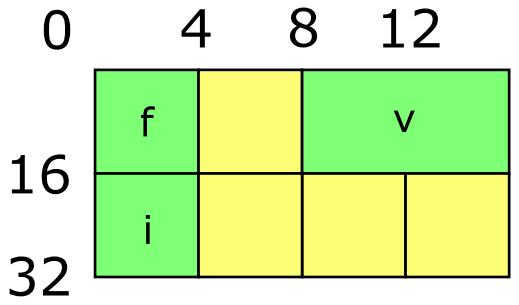


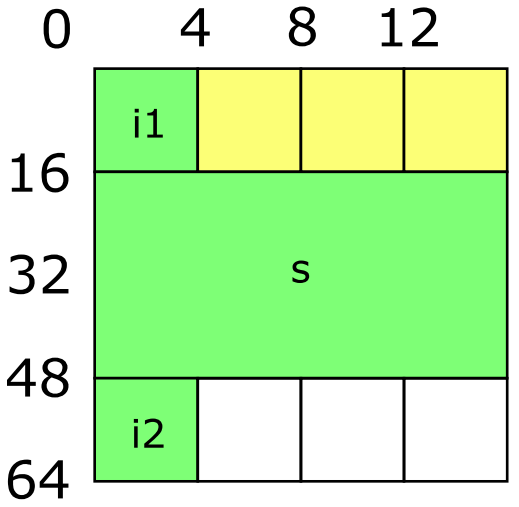
layout (std140)

- struktury: vel. „násobek(16) * n“, zarovnání 16

```
struct S
{
    float f;
    vec2 v;
    int i;
};

layout (std140) uniform U
{
    int i1;
    S s;
    int i2;
};
```





layout (std140)

- Rada na závěr: Dokud si nejste jistí, používejte pro všechno *vec4* a *mat4* :-)

Další možnosti layout

- Specifikace indexu vstupní proměnné VS

```
layout (location = 1) in vec4 normal;
```

- Specifikace indexu výstupní proměnné FS

```
layout (location = 0) out vec4 final_color;
```

- Specifikace binding pointu UBO

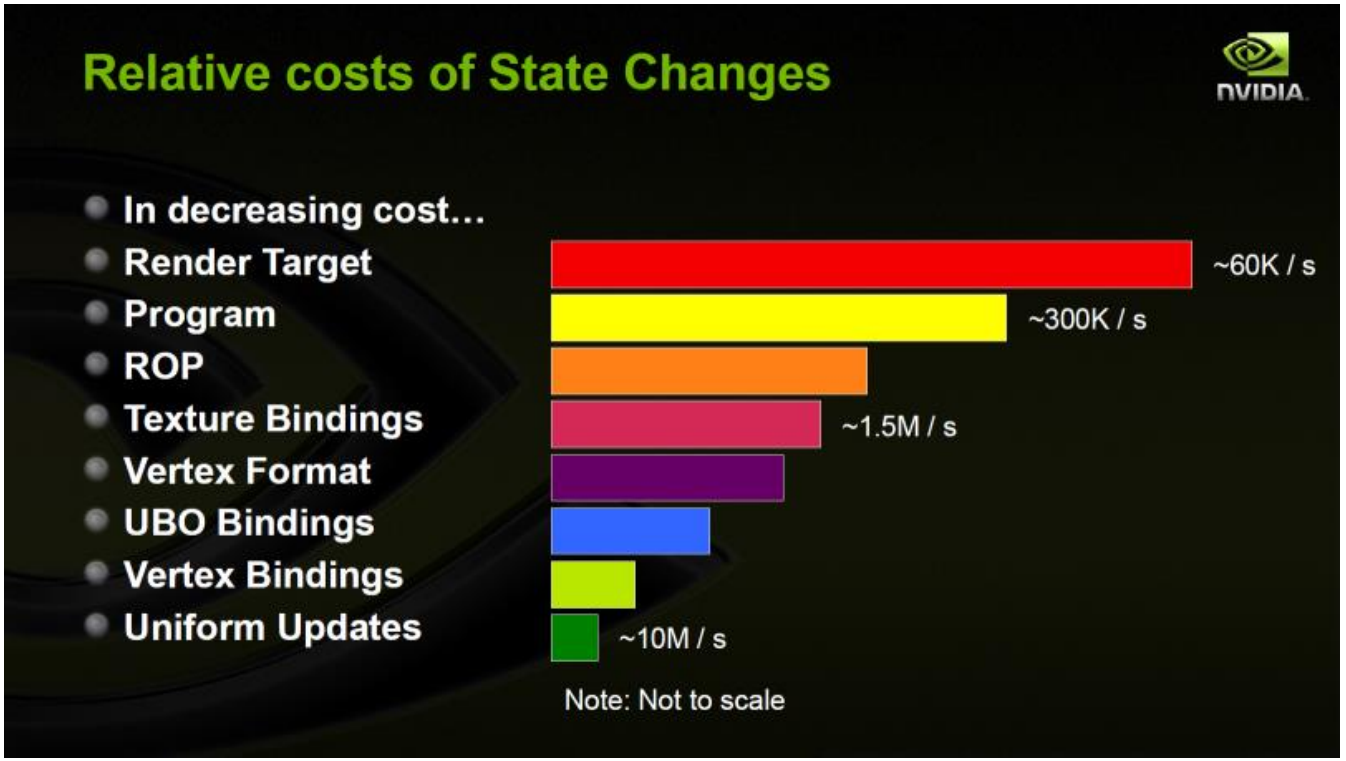
```
layout (std140, binding = 1) uniform LightData { ...
```

- Specifikace indexu uniformní proměnné
  - nelze sdílet mezi shadery

```
layout (location = 0) uniform int light_count;
```

Urychlování vykreslování

- Některé operace v OpenGL jsou náročnější
  - změny stavů a volání glDraw*



NVIDIA: Beyond Porting

Snižování počtu glDraw*

- Je lepší pomocí 1 glDraw* vykreslit 1000000 trojúhelníků, než pomocí 1000 glDraw* vykreslit 1000 trojúhelníků
- Instancování
- Další techniky

Instancování

- Kreslím naráz stovky objektů, které se liší jen několika málo atributy
  - pozice, barva apod.

GPU Gems 2

GPU Gems 3

Instancování

```
void glDrawArraysInstanced(GLenum mode,
    GLint first, GLsizei count,
    GLsizei primcount)
```

```
void glDrawElementsInstanced(GLenum mode,
    GLsizei count, GLenum type,
    const void *indices, GLsizei primcount)
```

- Stejné jako *glDrawArrays* a *glDrawElements*
- *primcount* = počet instancí

Instancování

- Ve vertex shaderu máme proměnnou *gl_InstanceID*, která obsahuje index dané instance
- Můžeme použít pro získání dat instance například z pole uloženého v UBO

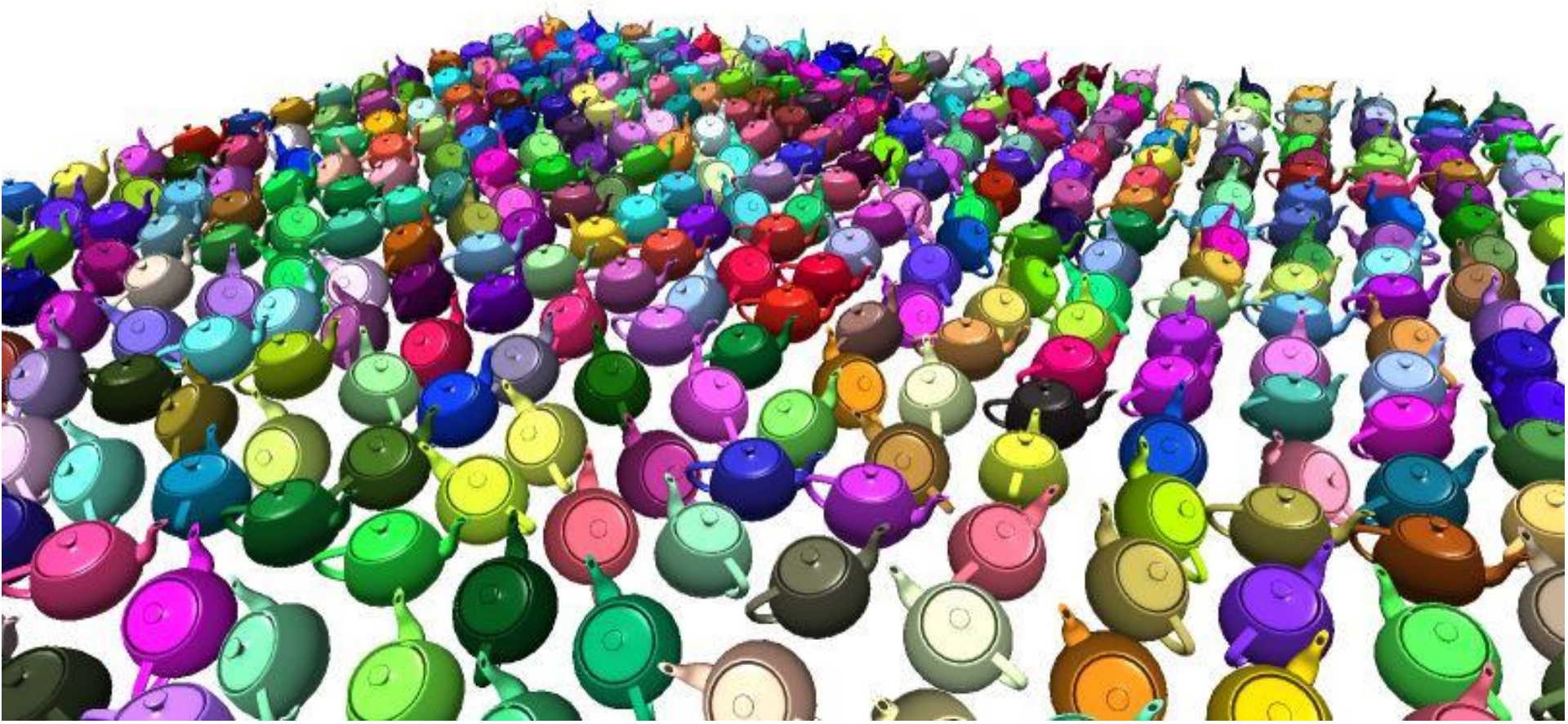
Instancování

```
uniform ObjectData
{
    mat4 model_matrices[400];
    vec4 model_colors[400];
};

out vec4 color;

void main()
{
    ...
    mat4 model_matrix = model_matrices[gl_InstanceID];
    color = model_colors[gl_InstanceID];
    ...
}
```

Instancování



Instancování

- Druhá možnost, jak poslat data instance, je přes VAO jako vstupní proměnnou VS

```
in mat4 model_matrix;
in vec4 model_color;

out vec4 color;

void main()
{
    ...
    // transform object using the model_matrix
    color = model_color;
    ...
}
```

Instancování

```
void glVertexAttribDivisor(
    GLuint index, GLuint divisor)
```

- *index* = index vstupní proměnné
- *divisor* = určuje, komu jsou data určena
  - 0 ... data pro každý vrchol (per-vertex)
  - 1 ... data pro každou instanci (per-instance)
  - 2 ... data pro každou druhou instanci
  - ...

Instancování

- Proměnné typu **mat** zadáváme jako několik samostatných vektorů
- Index vstupní proměnné je index prvního vektoru
- Další vektory mají index +1, +2, …

```
glBindBuffer(GL_ARRAY_BUFFER, model_matrices_vbo);

glEnableVertexAttribArray(model_matrix_loc);
glEnableVertexAttribArray(model_matrix_loc+1);
glEnableVertexAttribArray(model_matrix_loc+2);
glEnableVertexAttribArray(model_matrix_loc+3);

glVertexAttribPointer(model_matrix_loc, 4, GL_FLOAT,
   GL_FALSE, 16*sizeof(float), nullptr);
glVertexAttribPointer(model_matrix_loc+1, 4, GL_FLOAT,
   GL_FALSE, 16*sizeof(float), (void *)(4*sizeof(float)));
glVertexAttribPointer(model_matrix_loc+2, 4, GL_FLOAT,
   GL_FALSE, 16*sizeof(float), (void *)(8*sizeof(float)));
glVertexAttribPointer(model_matrix_loc+3, 4, GL_FLOAT,
   GL_FALSE, 16*sizeof(float), (void *)(12*sizeof(float)));

glVertexAttribDivisor(model_matrix_loc, 1);
glVertexAttribDivisor(model_matrix_loc+1, 1);
glVertexAttribDivisor(model_matrix_loc+2, 1);
glVertexAttribDivisor(model_matrix_loc+3, 1);
```

Příklady instancování

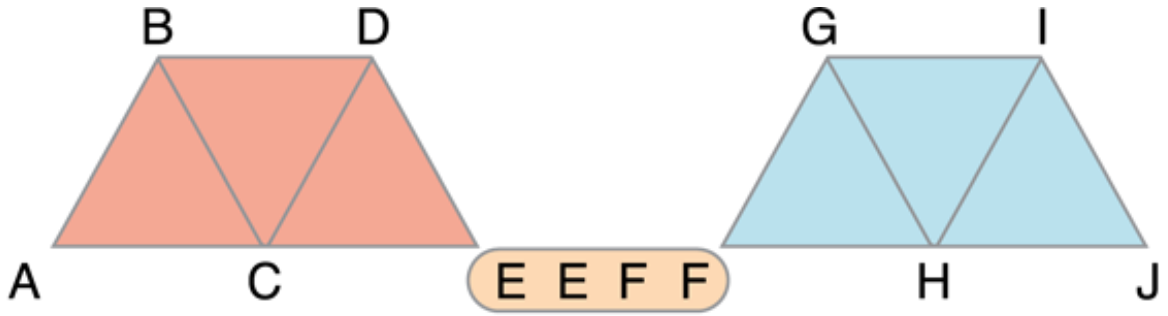
FIFA

Příklady instancování

Unreal Engine 4

Další techniky

- Degenerované trojúhelníky
  - OpenGL nekreslí trojúhelníky s nulovým obsahem, ani jako úsečky



developer.apple.com

Další techniky

- Primitive restart
  - Použitím specifického indexu OpenGL začne kreslit primitivum znovu
  - Je třeba povolit: *glEnable(GL_PRIMITIVE_RESTART)*
  - Je třeba zadat index: *glPrimitiveRestartIndex(idx)*
    - nejčastěji 0xffff pro *ushort* a 0xffffffff pro *uint*

```
0, 1, 2, 1, 3, 0, 65535, 4, 5, 6, 4, 7, ...
```

Další techniky

```
void glMultiDrawArrays(
 GLenum mode, const GLint *first,
 const GLsizei *count, GLsizei drawcount)
```

```
void glMultiDrawElements(
 GLenum mode, const GLsizei *count, GLenum type,
 const GLvoid * const * indices, GLsizei drawcount);
```

- Jako ne-Multi protějšky
- Vyžadují pole *first/count* a *count/indices* o velikosti *drawcount* namísto parametru
- Kreslí stejný druh primitiv (*mode* je stejný)

Další techniky

```
void glDrawElementsBaseVertex(
    GLenum mode, GLsizei count, GLenum type,
    const GLvoid *indices, GLint basevertex)
```

- Jako *glDrawElements*, ke všem indexům přičte *basevertex*
- Vhodné, pokud mám více geometrií ve stejných bufferech

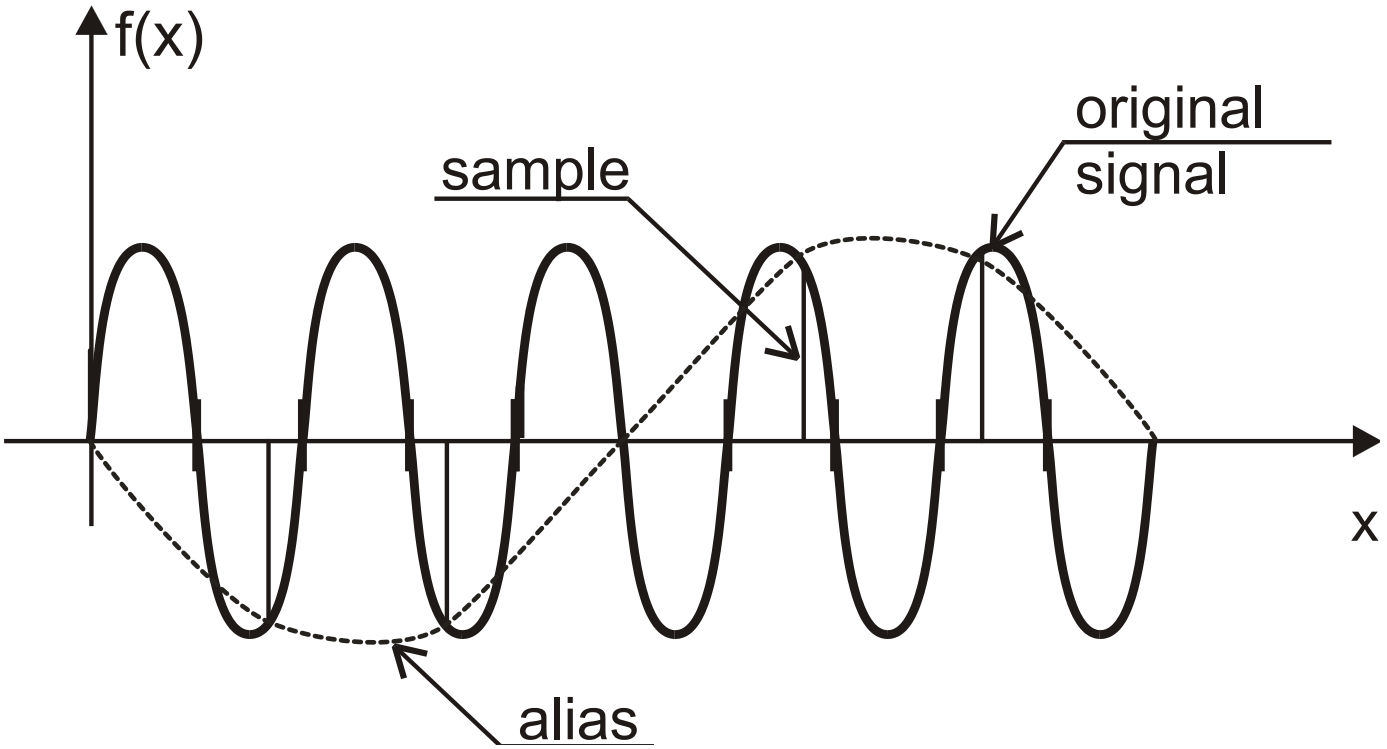
```
0, .. kostka .., 123, 0, .. konev .., 456, ...
```

Souhrn

- Instancování se hodí :-)
- Ostatní věci mějte v paměti, můžete se s nimi setkat
  - Ale ve svých projektech je pravděpodobně potřebovat nebudete.

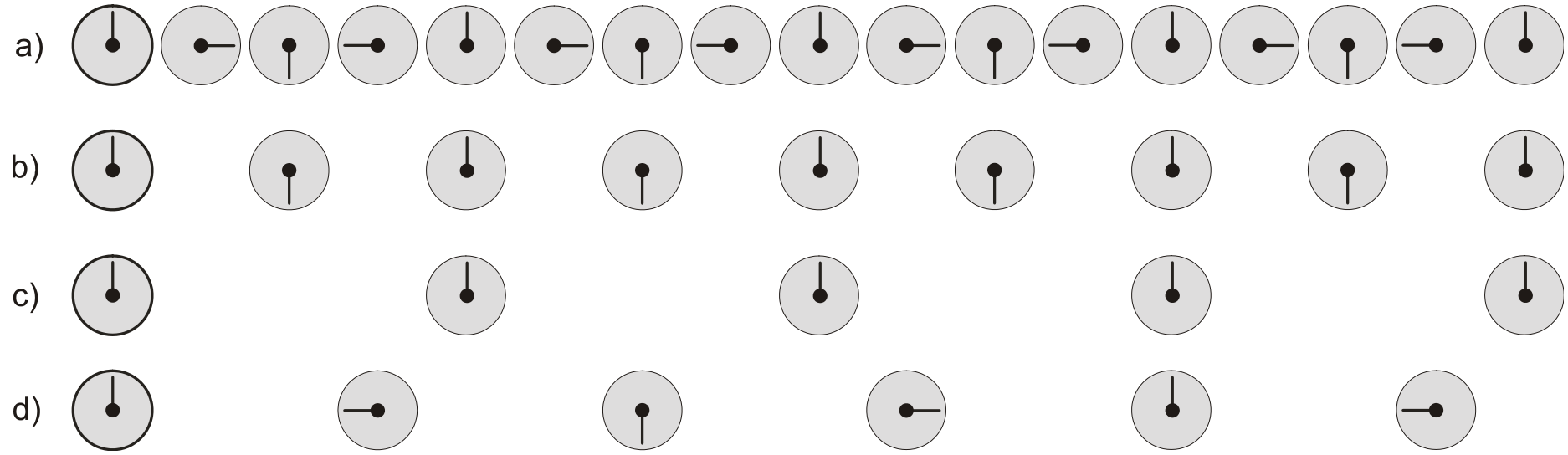
Aliasing

- Alias – „zubatost“ na okrajích
- Způsoben vysokofrekvenční informací navzorkovanou s nízkou frekvencí

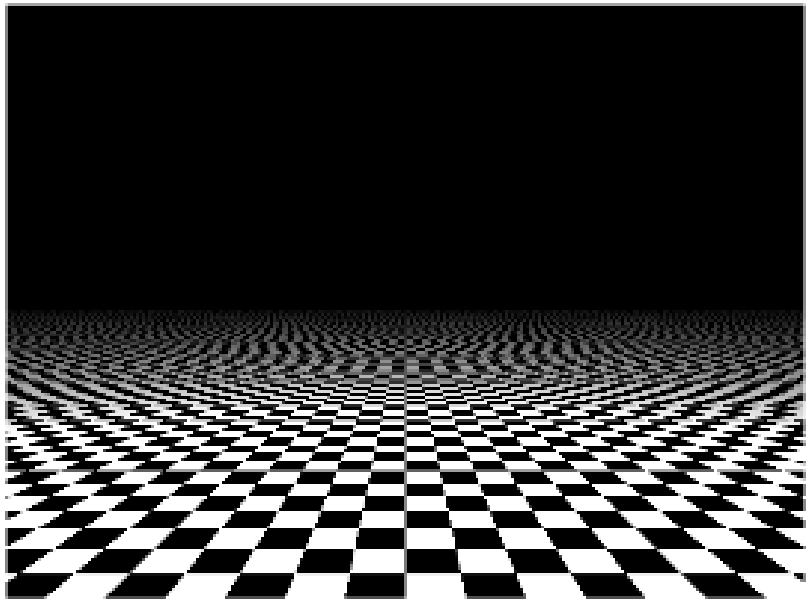


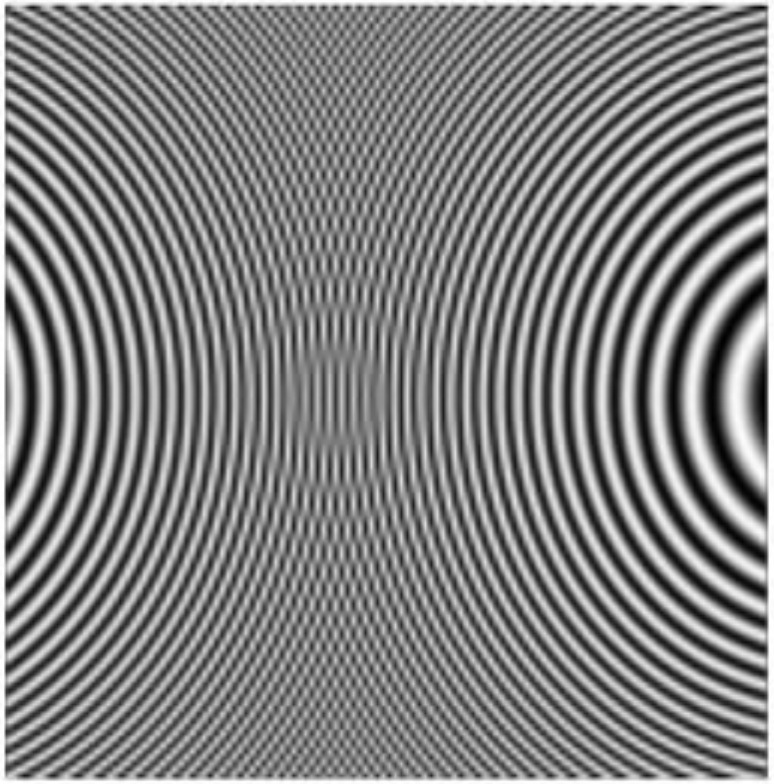
Aliasing

- Zubatost – jaggies
- Třepotání – flickering objects
- Efekt kol vagónu (aliasing v časové doméně)
- https://www.youtube.com/watch?v=jHS9JGkEOmA



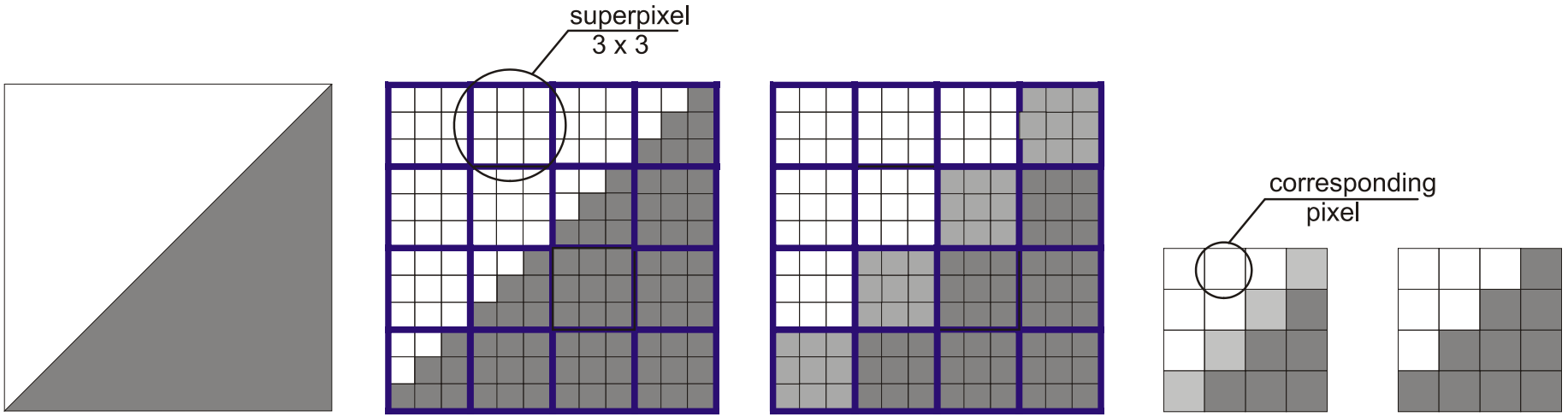
Aliasing - příklady





Antialiasing

- Antialiasing – odstranění/redukce aliasingu
  - postprocessing: rozmazání některých hran
    - Např: TXAA, FXAA, MLAA
  - supersampling

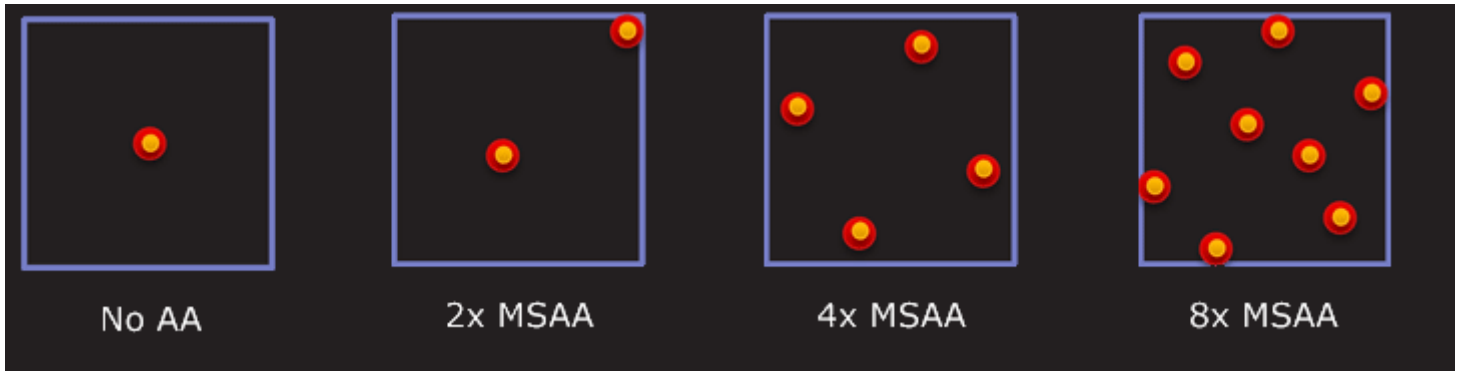


Supersampling

- Výhody:
  - jednoduchý na implementaci
  - kvalitní výsledky
- Nevýhody:
  - Vyžaduje velké množství paměti, bandwidth
    - FullHD: cca 2 mil pixelů, cca 16 MB x počet vzorků
  - Vyžaduje velký výpočetní výkon
- NVIDIA DSR (Dynamic Super Resolutions)

Multisampling

- Zjednodušený supersampling
- MSAA, např. 8x MSAA



anandtech.com

Multisampling

- Řeší problém výkonu, nikoliv paměti
- Fragment shader se spustí pouze jednou
  - barva všech fragmentů je stejná
  - výpočet hloubky a stencilu je prováděn pro každý fragment zvlášť
- Vyhladí okraje, nevyhladí vnitřek trojúhelníku
- Dobrý poměr cena/výkon

Multisampling – vytvoření FBO

- Vytvořit multisample textury do FBO

```
void glTexImage2DMultisample(GLenum target,
  GLsizei samples, GLenum internalformat,
  GLsizei width, GLsizei height,
  GLboolean fixedsamplelocations)
```

- *target* = *GL_TEXTURE_2D_MULTISAMPLE*
- *internalFormat, width, height* – glTexImage2D
- *samples* = počet vzorků
- *fixedsamplelocations* = stačí nám *GL_FALSE*

Multisample textury a FBO

- Všechny textury napojené na framebuffer objekt musí mít stejný počet samplů
  - Jinak máme nekompletní FBO
- Multisample textury nelze vzorkovat
  - lze speciálními funkcemi
- Nejpohodlnější je použít *glBlitFramebuffer* pro kopírování dat mezi multisample texturami/FBO a běžnými texturami/FBO

Multisampling – použití

- Povolení a zakázání
  - *glEnable/glDisable(GL_MULTISAMPLE)*
- Dotaz na počet vzorků
  - *glGetIntegerv(GL_SAMPLES, &samples)*
- Práce s děravými objekty
  - *glEnable/Disable(GL_SAMPLE_ALPHA_TO_COVERAGE)*
  - Převádí alphu fragmentu na pokrytí, provádí tak alpha test