Příklady postupů pro design aplikace

Grafické objekty

- Seznam vykreslovaných objektů
  - odvozené z jedné třídy
  - init(), update(), …
- Životnost objektu
  - startTime, endTime
- Priorita vykreslování
  - seřazení podle priority
    - např. vzdálenost
  - změna LOD od max. až k nule (nevykreslí se vůbec)
    - ev. LOD0=HQ, LOD++ → LQ
  - omezení na čas vykreslení

Časovače

- Jednoduchý časovač
  - obecný C (milisekundy) int clock()
  - sec. od inicializace glfw double glfwGetTime()
  - intervalový C++ std::chrono::steady_clock::now()
- HPET – High Precision Event Timer
  - tiky jádra → přesné, ale problematické
    - více jader, změna taktu CPU, …
  - assembler, podle modelu procesoru
- Vnější synchronizace
  - Obrazová data
    - snímky z videa, kamery, ...
  - Zvuková a hudební data
    - beats, samples, ...
  - Sdílený čas ze sítě
    - centrální synchonizace ze serveru
  - Vsync, ...

Vlákna

- 60 FPS = 16,7 ms
- Vykreslovat 3D lze jen v jednom vlákně
- Oddělit časově náročné operace
  - diskové operace
  - síťová komunikace
  - fyzika
  - zvuky (nejsou náročné, ale musí být přesné)
- Oddělit podle logiky
  - Model View Control apod.

Řízení zdrojů

- Nenahrávat ručně

```
tex1 = LoadTexture(„c:\textures\tex1.jpg“);
tex2 = LoadTexture(„c:\textures\tex2.jpg“);
tex3 = LoadTexture(„c:\textures\tex3.jpg“);
```

- Stovky textur, pro tisíce objektů
  - pomalé, náchylné na chyby
    - úniky paměti, neoptimální
- Existuje lepší způsob

Řízení zdrojů

- Samostatné manažery zdrojů
  - TextureManager, ShaderManager, SyncManager, …
  - Renderer.draw(scene,LOD)
  - singletony (jediná instance)
  - centralizované
  - lepší ladění, optimalizace
  - využití std::unordered_map apod.
    - rychlé vložení, vyhledání, smazání
      - asociativní pole, vyvážené binární stromy, …

Singleton

- Třída, ke které může existovat jen jediná instance

```
// MUST be at least C++11

class Singleton
{
public:
	static Singleton& getInstance()
	{
		static Singleton instance; // Guaranteed to be destroyed.
					   // Instantiated on first use = lazy init.
		return instance;
	}
	void doSomething(void) { std::cout << "Hi\n"; };
	int square(const int i) { return i * i; }

private:
	Singleton() {}                    // Hidden constructor

public:
	Singleton(Singleton const&) = delete;        // no copy constructor
	void operator=(Singleton const&) = delete;   // no assign operator
};


int main(int argc, char* argv[]) {

	// Accessing singleton methods:
	// a) calling getInstance
	Singleton::getInstance().doSomething();
	std::cout << Singleton::getInstance().square(5) << '\n';

	// b) create local variable with reference, can be global variable
	Singleton & smanager = Singleton::getInstance();

	smanager.doSomething();
	std::cout << smanager.square(5) << '\n';

}
```

příklad použití

- // v init() načíst všechno

  TextureManager.init(„data/textury“);

- // v draw() kódu

  TextureManager.bind(„pic1.jpg“, TEXTUREUNIT_1);
  ShaderManager.activate(„lights_tex_fog_noalpha“);
  ShaderManager.setUniform(„texunit“, 1);

  // nebo (pokud v TexM implementujeme LRU cache)

  texunit_with_pic1jpg = TextureManager.bind(„pic1.jpg“);
  ShaderManager.activate(„lights_tex_fog_noalpha“);
  ShaderManager.setUniform(„texunit“, texunit_with_pic1jpg);

Animace

- Změna polohy v čase
  - objekt
  - kamera
- Plynulá animace
  - 16 – 24 – 60 – 75 – 200 fps
  - záleží na
    - složitosti scény (kontrast)
    - rychlosti pohybu
- Cíl
  - vždy stejná rychlost
  - synchronizace se zvukem, hudbou
  - jednoduchý kód

Animace podle snímků

- V každém snímku

  pozice.x = pozice.x + posun; (všimni si: žádný čas)
- Výhody
  - vždy stejná trajektorie (benchmarky)
  - při malém posunu zaručen hladký pohyb
    - usnadňuje detekce kolizí
- Nevýhody
  - nelze zaručit hodnotu FPS
    - pokaždé jiný časový průběh, tj. rychlost (i na stejném počítači)
    - na novém HW nehratelně rychlé
  - nelze (jednoduše) synchronizovat pro síťové aplikace

Animace podle času – variabilní interval

- Založená na reálném nebo virtuálním čase
  - aproximuje fyzikální realitu
- V každém snímku

```
delta_t = old_time – time_now();
pozice.x = pozice.x + rychlost.x * delta_t;    //s = s0 + v*t
old_time = old_time + delta_t;
```

- Výhody
  - pohyb stejně rychlý nezávisle na FPS
  - snadná změna rychlosti (0.73 * clock())
  - možné vylepšit o zrychlení, odpor prostředí apod.
- Nevýhody
  - na pomalém HW velké skoky
    - vizuálně rušivé
    - znesnadňuje detekce kolizí
    - synchronizace přes síť (různé časové intervaly)

Animace podle času – konstantní interval

- Idea – nezávisle na vykreslování aktualizovat čas a fyzikální model
  - i několikrát v průběhu vykreslování jednoho snímku
- Výhody
  - pohyb stejně rychlý nezávisle na FPS
  - synchronizace přes síť
  - snadná změna rychlosti (0.73 * clock())
  - stabilní a jednoduchý model
    - snadná detekce kolizí, síťová synchronizace apod.
- Nevýhody
  - na pomalém HW velké skoky – vizuálně rušivé
  - obtížnější na programování – synchronizace, vlákna...

Synchronizace

- Interaktivní aplikace, hry
  - nutná rychlá reakce na uživatele
  - synchronizace vůči virtuální realitě a protihráčům
  - nastavitelné parametry kvality
    - ořezové roviny, textury, detailní modely, efekty, …
- Grafická dema, machinima
  - synchronizace vůči zvukové stopě
  - vynechávání snímků
  - zjednodušování scény – vynechávání objektů, LOD

Synchronizace

- Do SynManageru vkládáme události s parametry (čas atd.)
  - v daný čas se jednorázově vykonají, nebo se plynule mění, …
- Události

```
if ( ( time > 1000 ) || enemy.killed )
    akce();
```

- Deklarace

```
event die_fadeout {
    time = 10000;
    length = 3000;
}
```

- Použití
  - getValue() vrací hodnoty 0.0f – 1.0f

```
float pic_alpha = SyncManager.event(„die_fadeout“).get_value;
Renderer.drawPicture(„pic.jpg“, pic_alpha);
```

Plynulá změna parametrů

- Parametrická matematická funkce
  - lineární
  - exponenciální
  - spline, bezier, …
  - potřeba editor křivek apod. - např. ImGUI

```
float pic_alpha = SyncManager.curve(„fadeout“).eval(currentTime)
```

Tipy

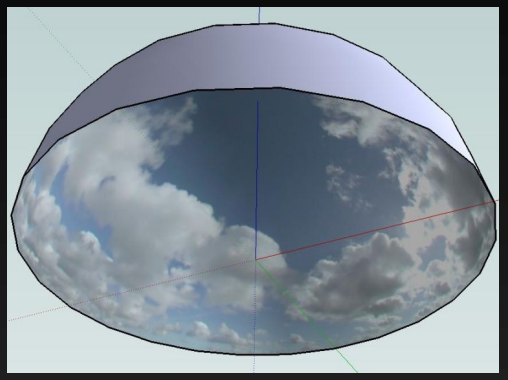
- Proměnné vše stejné (grafika, čas, apod.)
  - 0.0f … 1.0f
  - operace mezi 0…1 vychází obvykle v 0…1
- Jednoduchá inicializace aplikace
  - data scény v souboru, ne v kódu
  - JSON, XML, vlastní formáty, …
- Realtime modifikace
  - změna přes klávesy
  - pauza apod.

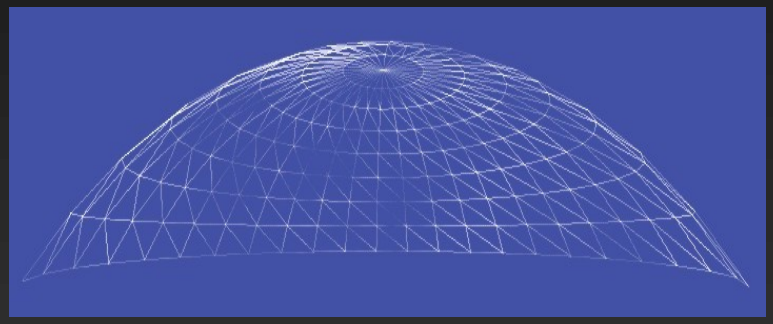
Tipy

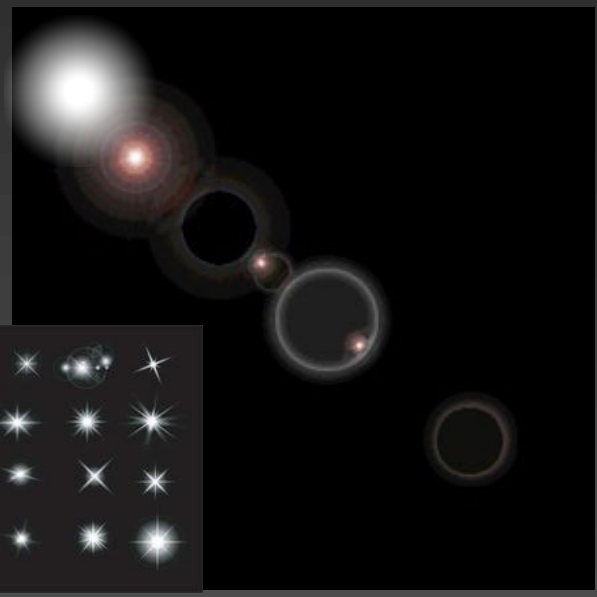
- Matematika
  - lineární algebra
    - matice, vektory
  - trigonometrie
  - numerické algoritmy – pro fyziku
  - křivky (splines, …)
  - ...

Tipy

- Zvýšení vizuální kvality
  - mlha
    - schovat FAR ořez do mlhy
  - obloha, mraky
    - polokoule (krychle) s texturou
    - polokoule (krychle) bez textury
    - předpočítaná nebo procedurální
  - slunce
    - obvykle není HDR
    - slunce + průhledná textura pro „glow“
    - směrový vektor + pohyblivé průhledné textury pro „lens flare“







Tipy

- Předpočítání všeho, co jde
- Rozčlenění scény kvůli přepočtům
  - statické objekty – dlaždice s terénem (čtverec, 6úhelník), …
  - pohyblivé objekty – hráč, střela atd.
- Rychlé
  - Postprocesing
    - rozmazání při rychlém pohybu (motion blur), mlha, …
  - změna barvy při zásahu, malém zdraví
    - průhledná textura, texturové operace, shadery, …
  - Antialiasing do úrovně 4
- Pomalé
  - Antialiasing (vysoké úrovně, anizo)
  - Víceprůchodové techniky (stíny, odlesky, DOF, …)
  - Obvykle příliš náročné