Gymnázium Jana Keplera

leden 2015
Jiří Oberfalzer

Maturitní práce

Předmět informatika

Dokumentace k projektu Treegen - generátor realistických modelů stromů

Vedoucí práce: Pavel Zbytovský

Gymnázium Jana Keplera leden 2015

Jiří Oberfalzer

Prohlašuji, že jsem jediným autorem této maturitní práce a všechny citace, použitá literatura a další zdroje jsou v práci uvedené.

Tímto dle zákona 121/2000 Sb.ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium Jana Keplera, Praha 6, Parléřova 2 oprávnění k výkonu práva na rozmožování díla(§ 13)a práva sdělování díla veřejnosti (§ 18)na dobu časově neomezenou a bez omezení územního rozsahu.

Gymnázium Jana Keplera

leden 2015
Jiří Oberfalzer

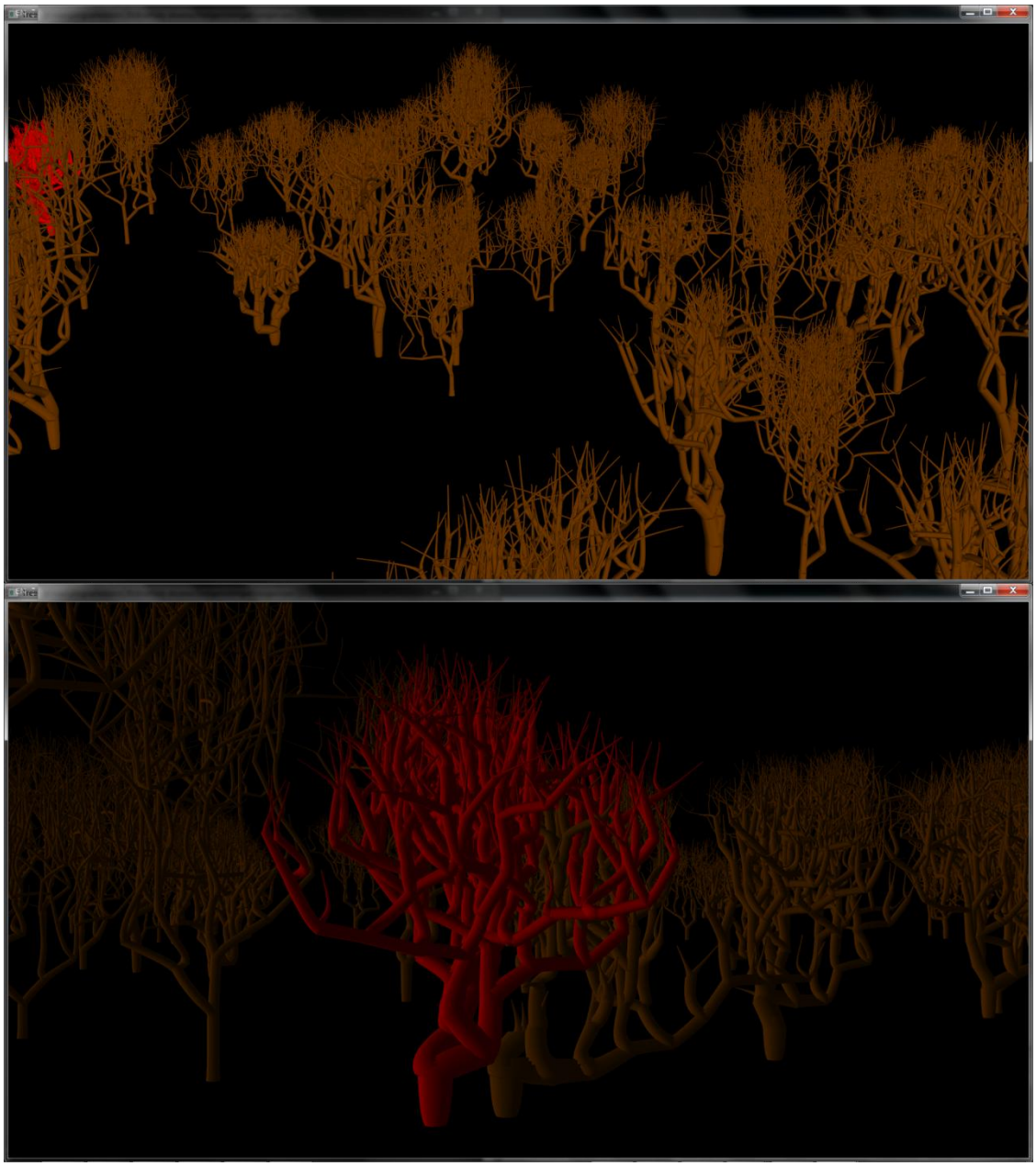
Dokumentace projektu Treegen - generátor realistických modelů stromů

Zadání

- Cílem je vytvořit generátor realistických modelů stromů, který bude z nastavených parametrů automaticky generovat texturu, vrcholy a stěny modelu.

Anotace

- Cílem mé práce bylo vytvořit software - generátor realistických modelů stromů, který bude snadno použitelný a dostupný pro uživatele se základní znalostí programování (amatéry, architekty, animátory apod.). Klíčovou vlastností programu je generování modelů stromů v reálném čase a degradace geometrické složitosti modelů. Inspiroval jsem se prací Josepha Penna a Jasona Webera: Creation and Rendering of Realistic Trees , z roku 1995. Systém je navržen tak, aby poskytl základní třídy pro práci s trojrozměrnou grafikou, případně byl snadno rozšiřitelný. Program je napsán v jazyce C++. Používal jsem online učebnici cplusplus.com. Pracoval jsem s knihovnami DirectX od Microsoftu pro vykreslování trojrozměrné grafiky a s opensource knihovnou Boost, z níž používám náhodné generátory čísel. Pracoval jsem také v IDE Code:Blocks. Používal jsem verzovací systém Git a hostitele Bitbucket. Pro kompilaci zdrojových kódu jsem dále zvolil Gnu Compiler Collection. Celý vývoj projektu probíhal pod systémem Windows XP. K vytvoření dokumentace projektu jsem použil značkovací jazyk Markdown, nástroj Pandoc pro jeho převod do různých formátů včetně docx a dále plaintextový editor Sublime. Domnívám se, že jsem plánovanému zadání z větší části vyhověl.



Spuštění programu

- Spuštění provedete následujícími příkazy pro git:
  - git init
  - git remote add r https://xmegapopcornz@bitbucket.org/xmegapopcornz/treegen.git
  - git pull r master
  - Spustitelný soubor je v podadresáři, pokud máte nainstalované DírectX půjde spustit.

- Nemáte-li DirectX, program vypíše chybu jako "V počítači chybí knihovna d3dx9_43" V tomto případě si stáhněte chybějící knihovny z branche knihovnyDx9 a vložte je k binárnímu souboru.
  - Knihovny stáhnete pomocí následujících příkazů pro git:
    - V adresáři projektu spusťte příkazovou řádku/powershell
    - git init
    - git remote add r https://xmegapopcornz@bitbucket.org/xmegapopcornz/treegen.git
    - git fetch r
    - git checkout knihovnyDx9

Kompilace projektu:

Budete potřebovat:

- TDM GCC 4.9 (64x)
- Boost 1.57.0
- Git
- knihovny DirectX9

Příkazy pro git:

- git init
- git remote add r0 https://xmegapopcornz@bitbucket.org/xmegapopcornz/treegen.git
- git pull r0 master

Úprava kompiluj.bat

- V souboru kompiluj.bat v adresáři projektu změňte hodnotu proměnné CESTA_KOMPILATORU na složku s instalací TDM GCC
- Nyní je vše připraveno.
  - spusťte soubor kompiluj.bat
  - spusťte bin/mingwTreegen.exe

Instalace prostředí:

Budete potřebovat:

- Codeblocks
- TDM GCC 4.9 (64x)
- Boost 1.57.0
- Git

Příkazy pro git:

- git init

- git remote add r0 https://xmegapopcornz@bitbucket.org/xmegapopcornz/treegen.git
- git fetch r0
- git checkout kmen

Další kroky:

- V Codeblocks nastavte nový kompilátor:
  - Settings -> Compiler -> v selected compiler vyberte položku GNU GCC Compiler -> Copy -> zadat název nového kompilátoru, např.: "Twilight Dragon Media compiler"
  - přejděte na záložku Toolchain executables
  - Compiler´s installation directory nastavte na adresář kompilátoru obsahující podsložku "bin"
  - C++ compiler nastavte na: "x86_64-w64-mingw32-g++.exe"
  - Linker pro dyn. knihovny na totéž
- Nastavte projektu Build options:
  - Project -> Build options
  - Vlevo vyberte target "tdmDebug"
  - Vyberte nově přidaný kompilátor "Twilight Dragon Media compiler"
  - V záložce "Compiler settings" -> "other options" -> přidejte "-g -std=gnu++1y"
  - V záložce "Linker settings" by již mělo být "d3d9, d3dx9_43, dinput8, dxguid"
  - V záložce "Search directories" -> "compiler" -> add -> najděte cestu kořenového adresáře boostu, pozor cesta nesmí končit lomítkem!
- Výsledný Build log by měl vypadat nějak takto:

```
-------------- Build: tdmDebug in mingwTreegen (compiler: Twilight Dragon
Media compiler)---------------

x86_64-w64-mingw32-g++.exe -g -std=gnu++1y -I"H:\devpaks\boost_1_57_0" -c
"H:\Treegen\Engine.cpp" -o .objs\Engine.o
x86_64-w64-mingw32-g++.exe -g -std=gnu++1y -
I"H:\Treegen\devpaks\boost_1_57_0" -c "H:\Treegen\Usecka.cpp" -o
.objs\Usecka.o
x86_64-w64-mingw32-g++.exe  -o mingwTreegen.exe .objs\Engine.o .objs\Helper.o
.objs\Input.o .objs\Kamera.o .objs\Kontroler3d.o .objs\main.o .objs\msgProc.o
.objs\SWU\Konzole.o .objs\SWU\Okno.o .objs\Tree.o .objs\Tree3.o .objs\Tvar.o
.objs\Usecka.o  -static  -ld3d9 -ld3dx9_43 -ldinput8 -ldxguid
Output file is mingwTreegen.exe with size 4.33 MB
Process terminated with status 0 (0 minute(s), 17 second(s))
0 error(s), 0 warning(s) (0 minute(s), 17 second(s))
```

Ovládáni:

- w, s, a, d - dopředu, dozadu, doleva, doprava

- q, e - nahoru, dolu
- ↑, ↓, ←, → - pohled nahoru, dolu, vlevo, vpravo
- myš - směr kamery
- levé tlačítko myši - přidá strom, náhodně vygenerovaný v reálném čase
- příkazy:
  - Pro zadání příkazu stiskněte mezerník+příkaz[+mezera+argument]+enter. Argument je celé číslo. Není-li specifikován příkaz, program předpokládá nastavení rychlosti.
  - "testy" - spustí testy programu, které otestují všechny možné příkazy
  - "jeden" - vymaže geometrii scény a vygeneruje jeden strom.
  - "rychl" - nastaví rychlost pohybu kamery, výchozí rychlost je 5000
  - "odebe" - odebere posledně přidaný strom
  - "gener" / "pride" - přidá zadaný počet stromů, je-li menší než 200
  - "eleme" + 0(bod), 1(přímka), 2(plocha) - tam, kde je to možné, změní vykreslovaný element na zvolený element
  - "backc" - prohodí barvu pozadí mezi černou/bílou
  - "vymaz" - vymaže všechnu geometrii na scéně
  - "regen" - znovu vygeneruje počáteční scénu
  - "reset" - obnoví počáteční podmínky programu
  - "fovde" + číslo z <2, 50) - změní horizontální zorný úhel, číslo je jmenovatelem pro úhel v radiánech(PI/x)
  - "citli" + číslo z <0, 10> - změní citlivost myši
  - "wiref" - prohodí render state mezi solid/wireframe
  - "osvet" - vypne/zapne osvětlování
  - "nerot" - zastaví rotaci všech modelů
  - "zhasn" - zhasne baterku(světlo ve směru hledí)
  - "rozzn" - rozžne baterku
  - "shadi" - prohodí způsob stínování Goraud/flat
  - "culli" - vypne/zapne culling(technika, kdy se nevykreslují skryté trojúhelníky)

Programátorská dokumentace

vstupní bod programu: main

1. Inicializace statických proměnných pocetInstanciStromu třídy

```
#include <windows.h>
#include "Engine.hpp"
#include "globals.hpp"

int t::Tree::pocetInstanciStromu = 0;   // inicializace staticke promenne
tridy Tree, slouzi k uchovani poctu strukturovych stromu. Inicializace
statickych promennych je nutne provest vzdy mimo tridu.
```

```
int t3::Tree3::pocetInstanciStromu = 0;     // inicializace staticke promenne tridy Tree3, slouzi k uchovani poctu 3D stromu
int se::Svetlo::pocetInstanciSvetla = 0;       // inicializace staticke promenne tridy Svetlo, slouzi k uchovani poctu svetel sceny

int main (HINSTANCE hThisInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR lpszArgument,
                    int nCmdShow)
{
    sk::Konzole iKonzole;       // inicializace  instance tridy Konzole, ktera se stara o okno konzole, vstup a vystup konzole
    sw::Pozice iPozice {0, 0};      // inicializace objektu Pozice, ktery nese informace o pozici okna
    sw::Rozmery iRozmery {windowWidth, windowHeight};
    sw::Okno iOkno(iPozice, "Tree", hThisInstance);    // inicializace objektu tridy Okno, ktery ma za ukol starat se o okno Win programu, jeho presouvani do popredi a zpravy z systemu
    se::Engine iEngine(&iOkno);         // inicializace objektu tridy Engine, jehoz funkce potrebuji(zejmena kvuli DX) pristup k informacim o oknu
    iEngine.dejKonzoli(iKonzole);           // predani reference na iKonzoli objektu iEngine, aby mohl pouzivat jeji funkce
    iEngine.priprav();    // Pripravi rozhrani DirectX, vyhradi pristup ke graficke karte, nastavi BackBuffer, Depth stencil, svetla, Render state, vytvori geometrii stromu
    iOkno.ukaz();       // ukaze okno
    while ( iOkno.jeOtevrene() ) {              // smycka programu
        iEngine.prectiVstupAUpravKameru();      // zavola metodu Kontroleru3d iKontroler3d.prectiVstupAUpravKameru(arg), ktera aktualizuje matice View podle uzivatelskeho vstupu
        if(!iOkno.postarejSeOZpravy()){                     // Zmacknul-li uzivatel krizek, ukonci program.
            iEngine.releaseD3d();               // uvolni pamet alokovanou v prubehu programu, uvolni pristup ke GP, klavesnici a mysi
            break;
        }
        else if(iEngine.prikaz == "odejdi" ){               // , v pripade zpravy odejdi ukonci program (prave tlacitko mysi, nebo prikaz)
            break;
        }
        iEngine.render3d();                     // vykresli scenu, neboli prohodi back a front buffer, predchozi obsah backbuffru je smazan
    }
    return 0;
}
```

třída Engine

- Instance této třídy lze ovládat uživatelským vstupem z příkazové řádky(CLI) za pomocí member function pointrů. Popíšu nyní způsob realizace.

- Nejprve pro zjednodušení deklaruji typ UkazatelNaFunkciEnginu(se::Engine member function pointer funkce beroucí za parametr jednu proměnnou typu float a nic nevracející) a typ mapy std::string a UkazatelNaFunkciEnginu:

```
class Engine;   // nejprve je nutne preddefinovat tridu Engine, aby bylo mozne nasledujici
using UkazatelNaFunkciEnginu = void (se::Engine::*)(float);
using MapaFunkciEnginu = std::map<std::string, UkazatelNaFunkciEnginu>;
```

- Poté si v hlavičce třídy Engine deklaruji proměnnou(instanci mapy std::string a UkazatelNaFunkciEnginu) mapaFci typu MapaFunkciEnginu:

```
MapaFunkciEnginu mapaFci;
```

- Nyní je třeba definovat member funkci Engine, která bere za parametr název funkce(std::string), prohledá mapu mapaFci a vrátí ukazatel na požadovanou member funkci Enginu:

```
UkazatelNaFunkciEnginu Engine::najdiFunkci(const std::string& nazevFce)
// tato funkce nesmi menit nazevFce, proto je const&
{
    MapaFunkciEnginu::const_iterator iIteratoruMapyFunkci;      // deklarace iteratoru objeku Pair mapy
    iIteratoruMapyFunkci = mapaFci.find(nazevFce);              // do iteratoru se ulozi ukazatel na Pair obsahujici nazevFce ktery najde funkce find(std::string) v mape
    if(iIteratoruMapyFunkci == mapaFci.end())                   // pokud je v iteratoru ukazatel na konec mapy(jeste neexistujici Pair), znamena to, ze funkce find() nenasla odpovidajici Pair
        throw EngineVyjimka("nenalezena pozadovana funkce enginu");     // program vypise vyjimku
    else
        return iIteratoruMapyFunkci->second;                    // funkce byla nalezena, vratim tedy Engine member function pointer, coz je druhy objekt v Pair
}
```

- Dále member funkci Engine, která bere za parametr ukazatel na požadovanou member funkci Enginu a parametr pro požadovanou funkci a zavolá ji.

```
void Engine::zavolejFunkci(UkazatelNaFunkciEnginu ukazatelFce, const float& arg)
{
    (this->*ukazatelFce)(arg);  // timto zpusobem zavolam na instanci se::Engine funkci, na kterou ukazuje Engine member function pointer ulozeny v parametru ukazatelFce a které postoupím parametr arg
}
```

- V tuto chvíli už jen inicializuji mapaFci (v konstruktoru Engine)

```
mapaFci["gener"] = &Engine::pridejStrom;
mapaFci["eleme"] = &Engine::setRenderElement;
```

```
mapaFci["backc"] = &Engine::switchClearColor;
mapaFci["vymaz"] = &Engine::odeberStromy;
mapaFci["regen"] = &Engine::regenerujStromy;
mapaFci["reset"] = &Engine::reset;
mapaFci["fovde"] = &Engine::dejFov;
mapaFci["citli"] = &Engine::dejCitlivost;
mapaFci["wiref"] = &Engine::switchWireframe;
mapaFci["osvet"] = &Engine::switchOsvetlovat;
mapaFci["nerot"] = &Engine::nerotuj;
mapaFci["zhasn"] = &Engine::zhasniSvitilnu;
mapaFci["rozzn"] = &Engine::rozzniSvitilnu;
mapaFci["shadi"] = &Engine::switchShading;
mapaFci["culli"] = &Engine::switchCulling;
mapaFci["pride"] = &Engine::pridejStrom;
```

- Teď již mohu volat fce, které si uživatel přeje:

```
zavolejFunkci(najdiFunkci(prikaz), arg);
```

třída Kontroler3d

- Třída Kontroler3d plní funkci kontroleru v návrhu Model, View, Controler, tedy se stará o uživatelský vstup prostřednictvím třídy Input(hlavní ovládání simulace klávesnicí a myší) a Konzole(ovládání simulace příkazy). Tento návrh má za cíl co největší oddělení částí kódu obstarávajících View, Controler, Model. Měly by být nezávislé, aby změna kódu jednoho, nevyžadovala změnu ostatních.

```
#ifndef __H_KONTROLER3D_INCLUDED_       // automaticke ohlidani vlozeni teto
hlavicky, aby nedoslo k redefinicim apod.
#define __H_KONTROLER3D_INCLUDED_

#include <windows.h>
#include "Input.hpp"        // vložím hlavičku třídy Input
#include "Kamera.hpp"       // vložím hlavičku třídy Kamera
#include "SWU/Konzole.hpp"  // vložím hlavičku třídy Konzole

namespace sktrl
{


class Kontroler3d
{
public:
    Kontroler3d( HWND& );   // konstruktor inicializje instance trid Kamera a
Input
    ~Kontroler3d();     // destruktor odpoji pointer matrixView od
iKamera.g_View,
    std::string& prectiVstupAUpravKameru(float&);   // tato funkce
aktualizuje stav iInput, podle stisknutych klaves zavola odpovidajici funkce
upravujici view, je volana v iEngine a obsahuje pro nej prikazy
    void posunKameru(skam::Smer kam, float nasobitel)      // posune kameru o
```

```
dany usek
    void otocKameru(skam::Smer kam);        // otoci kameru podle citlivosti
urcitym smerem
    void nastavRychlost(float rychlost);        // nastavi rychlost posunu
kamery
    void nastavCitlivost(int citlivost);        // nastavi citlivost rotace
kamery
    void dejKonzoli(const sk::Konzole& x) { iKonzole = x;}      // umoznuje
pristup k funkcim konzole
    void dejMoznePrikazy( std::string* _moznePrikazy, int pocet );      //
pole prikazu, ktere nastavuje Engine
    D3DXMATRIX * vemView();     // zajistuje pristup k matici View, ktery
potrebuje Engine pro funkci vykresli()
    const D3DXVECTOR3& vemSmerHledi();      // vrati referenci na vektor
smeru hledi, ktery nelze menit
    const D3DXVECTOR3& vemUmisteni();       // vrati const referenci na
vektor pozice hledi
private:
    HWND hWnd;      // nese informaci o window handle prislusiciho Windows
okna
    si::Input iInput;       // deklarace instance iInput tridy Input,
starajici se o syrovy vstup
    sk::Konzole iKonzole;       // deklarace instance iKonzole tridy Konzole,
aby bylo mozne vyuzivat ruzne funkce upravujici vystup apod.
    skam::Kamera iKamera;       // deklarace instance iKamera tridy Kamera v
namespace skam, ktera se stara o matici pohledu a efekty kamery
    D3DXMATRIX * matrixView;        // pointer na matici view iKamery
    std::string stav;           // stav vstupu obsahujici zpravy a prikazy
pro Engine
};


}
#endif // __H_KONTROLER3D_INCLUDED_
```

třída t::Tree3

- Tato třída má za úkol definovat strom jak abstraktně(pomocí definice způsobu generování vlastností jednotlivých větví a celkové struktury reprezentované binárním polem, které definuje jaké větve budou mít potomky), tak geometricky pomocí pravidelných n-bokých hranolů(technicky jsou definované vrcholy a indiciemi), jejichž konce jsou propojeny. Také dědí po abstraktní třídě TvarDx funkce a objekty, jejichž prostřednictvím je geometrie vykreslena v se::Engine.
  Dále třída obsahuje dvě dynamicky alokovaná pole std::vector<>. Jedno obsahuje instance třídy VrcholBK, v nichž je uložena pozice vrcholu a jeho normála(důležitá pro osvětlení). Druhé je typu long a obsahuje jednotlivé indicie, které po trojicích definují jednotlivé trojúhelníkové plošky, jejichž pixely se v grafické kartě zbarvují interpolací normál mezi vrcholy, grafická karta pak dopočítá množství světla dopadajícího na plátno a z toho rasterizer barvu jednotlivých pixelů. Existuje více způsobů stínování,

DirectX9 umožňuje přepnout mezi flat stínováním(pixel shader nic neinterpoluje, nastaví celé plošce trojúhelníku stejnou barvu v závislosti na velikosti zetová složky normály vrcholu kolmé k povrchu) a stínováním Goraud(pixel shader počítá barvu na základě lineární interpolace normál mezi vrcholy)

```
class Tree3
{
public:
    static int pocetInstanciStromu;       // pocet stromu

    Tree3();        // defaultni konstruktor nic neinicializuje
    Tree3(t::DruhStromu&, D3DXMATRIX& pocatek, LPDIRECT3DDEVICE9* _pzarizeni,
float zRot);      // konstruktor vola hlavni funkce pro generovani vsech
technickych prvku
    Tree3(Tree3&& tmp);                        // move ctor se pouziva pri
realokaci dynamickeho pole vector<>
    Tree3& operator= (Tree3&&);                // move assignment
    ~Tree3();       // destruktor dealokuje low-level pole, paIndicie,
cstmvtxVrcholy a vlastnostiVetvi, dale dyn. alok. pointry: bufferVrcholu a
bufferIndicii

    void aktualizujMatici();        // provede operace rotace a posunu na
maticich
    void vykresli(bool osvetlovat) const;       // preda vsechna data GP a
pomoci rozhrani DX(fce DrawIndexedPrimitive()) vykresli vsechny plosky
    void nastavVykreslovaciElement(t::Element _f);         // prenastavi typ
vykreslovanych primitiv
    void nastavRotaci(float _f);          // nastavi modelu rotaci kolem
vertikalni osy

private:
    sk::Konzole iKonzole;
    D3DMATERIAL9 material;                // material stromu obsahuje barvu
pro ruzne druhy osvetleni a muze obsahovat texturu ktera se ma pouzit
    t::DruhStromu druhStromu;             // druh stromu obsahuje informace,
ktery postup pouzit pro generovani vetvi, dale kolik generaci vetvi
vygenerovat, jaka je pravdepodobnost rozvetveni, kolika-boke hranoly pouzit a
kolik nejvyse jich ma byt na jednu vetev
    VlastnostiVetve* vlastnostiVetvi;     // abstraktni parametry vetvi
    std::vector<se::Usecka> kolmice;      // pomocne kolmice, graficke
reprezentace normal
    VrcholBK* cstmvtxVrcholy;
    long* paIndicie;
    std::vector<VrcholBK> vrcholy;        // dynamicky container pro vrcholy
    std::vector<long> indicie;            // dynamicky container pro indicie
    LPDIRECT3DVERTEXBUFFER9* bufferVrcholu;     // buffer s raw daty vrcholu
pro GP
    LPDIRECT3DINDEXBUFFER9* bufferIndicii;      // buffer s raw daty indicii
pro GP
```

```
    bool generujVlastnostiVetvi();      // vygeneruje vlastnosti vetvi podle
strukturniho klice, vola generujVlastnostiVetve()
    VlastnostiVetve generujVlastnostiVetve( const VlastnostiVetve& parent,
int strana, t::DruhStromu& _tType );     // vygeneruje a vrati novy soubor
vlastnosti vetve do instance tridy VlastnostiVetve
    bool vytvorBufferVrcholu();            // alokuje misto pro vrcholy v
graficke paměti a vytvoří vertex buffer do bufferVrcholu
    bool vytvorBufferIndicii();             // alokuje misto pro indicie v
graficke paměti a vytvoří index buffer do bufferIndicii
    void znicBuffery();                     // na konci existence instance
tridy je treba uvolnit vyhrazene misto v graficke pameti
    void znicOstatniPointry();              // dealokuje pamet vyhrazenou pro
vlastnostiVetvi, paIndicie, cstmvtxVrcholy z heap
    bool uzamkniPoleDoBuffru();       // vyhradi pristup pro zapisovani do
graficke pameti
    bool generujVykreslovaciDataVetvi();     // generuje vrcholy, indicie,
normaly stromu do vector containeru vrcholy, indicie
    bool generujElementyVetve( VlastnostiVetve& );      // generuje clanky
jedne vetve a indicie
    void generujVrcholyKruhu( const D3DXVECTOR3& pocatek, VlastnostiVetve&
pV, float r, float radiusZ, float sklony, float sklonz, float Dens );
// generuje vrcholy jednoho mnohouhelnika vetve do vectoru vrcholy
    void generujVrcholyVetve( const D3DXVECTOR3& pocatek, int kolikClanku,
VlastnostiVetve& pV );       // generuje vrcholy vetve, vola
generujVrcholyKruhu
    int generujIndicieVetve( int cislo, int kolikClanku );      // generuje
indicie vetve, vola generujIndicieKruhuXY
    void generujIndicieKruhuXY( int x, int y );         // generuje indicie
kruhu cislo x, y, vola generujIndicieElementu
    void generujIndicieElementu( t::Element e, int );       // prida par
indicii(druha je ++prvni) do vector containeru indicie
    void generujIndicieElementu( t::Element e, int, int );      // prida par
indicie elementu do vector containeru indicie
    void pridejNormaluVrcholu( int pozice, D3DXVECTOR3 kolmice );       //
prida normalu kolmice vrcholu cislo pozice
    D3DXVECTOR3 spocitejNormaluVrcholu( int a, int b, int c );          //
spocte a vrati normalu plosky tvorene vrcholy cislo a, b, c, vyuziva jejich
pozic ve vector containeru vrcholy
    D3DXVECTOR3 spocitejNormaluVrcholu( int a );    // spocte a vrati normalu
vrcholu cislo a, kterazto je souctem normal prilehlych plosek
    bool odemkniVrcholyProCteni();      // sdeli graficke karte, ze muze
zacit vykreslovat buffry, ztracim tim pravo zapisovat do techto buffru
    bool zkopirujVrcholyDoBuffru( const std::vector<VrcholBK>& _vrcholy );
// Vola vytvorBuffer, zkopiruje vrcholy z dynamickeho containeru vector do
klasickeho c pointer pole cstmvtxVrcholy
    bool zkopirujIndicieDoBuffru( const std::vector<long>& _indicie );   //
Vola vytvorBuffer, zkopiruje indicie z dynamickeho containeru _indicie do
klasickeho pole paIndicie
};
```

- konstruktor stromu:
    1. Zavolá funkci generujVlastnostiVetvi(), která
        - vygeneruje ternární pole obsahující strukturu stromu podle zadaných parametrů urovenRozvetveni(počet generací větví), pravdepodobnostRozdvojeni v instanci třídy t::DruhStromu
        - spočítá počet větví
        - inicializuje vlastnosti první větve(kmenu) do vlastnostiVetvi[0]
        - inicializuje vlastnosti zbylých větví
    2. Zavolá funkci generujVykreslovaciDataVetvi(), která
        - vygeneruje vrcholy stromu
        - vygeneruje indicie stromu
        - nakonec vygeneruje normály všem vrcholům
    3. Spočítá vrcholy, indicie a ostatní elementy
    4. Zavolá funkci uzamkniPoleDoBuffru(), která
        - Zavolá funkci vytvorBufferVrcholu(), jenž alokuje místo pro vrcholy v grafické paměti a vytvoří buffer vrcholů
        - Vyhradí programu zapisovací přístup do buffru prostřednictvím pole cstmvtxVrcholy, čímž sdělí grafické kartě, aby tento buffer zatím nepoužívala
        - Zavolá funkci vytvorBufferIndicii(), jenž alokuje místo pro indicie v grafické paměti a vytvoří buffer indicií
        - Vyhradí programu zapisovací přístup od buffru prostřednictvím pole paIndicií, čímž sdělí grafické kartě, aby tento buffer zatím nepoužívala
    5. Zavolá funkci zkopirujVrcholyDoBuffru(), které předá jako argument dynamický container vector vrcholy, tato funkce
        - Zapíše do grafické paměti prostřednictvím jednoduchého pole cstmvtxVrcholy provázaného s vertex buffrem data z dynamického containeru vrcholy
    6. Zavolá funkci zkopirujIndicieDoBuffru(), které předá jako argument dynamický container vector indicie, tato funkce
        - Zapíše do grafické paměti prostřednictvím jednoduchého pole paIndicie provázaného s index buffrem data z dynamického containeru indicie
    7. Zavolá funkci odemkniVrcholyProCteni(), která
        - Sdělí grafické kartě, že již nechci do buffrů zapisovat a že je již žádoucí, aby je použila pro vykreslování.
- kód konstruktoru

```
try {
    if(!generujVlastnostiVetvi()) throw StromVyjimka("Nepodarilo se
vygenerovat vlastnosti vetvi.");   // generuje vlastnosti vetvi
    if(!generujVykreslovaciDataVetvi()) throw StromVyjimka("Nepodarilo se
vygenerovat vrcholy, normaly nebo indicie vetvi.");    // vygeneruje vrcholy,
normaly a indicie stromu
    pocetVrcholu = spoctiVrcholy();     // spocita vrcholy, jejichz presny
pocet je treba znat pro vytvoreni buffru vrcholu a vykresleni
    pocetElementu = spoctiElementy();       // spocita clanky, trojuhelniky a
indicie, jejichz pocet je treba znat pro vytvoreni buffru indicii a spravne
vykresleni
    if(!uzamkniPoleDoBuffru()) throw StromVyjimka("Nepodarilo se uzamknout
vrcholy do buffru.");    // aby bylo
    if(!zkopirujVrcholyDoBuffru(vrcholy)) throw StromVyjimka("Nepodarilo se
zkopirovat vrcholy do statickeho pole a buffru.");
    if(!zkopirujIndicieDoBuffru(indicie)) throw StromVyjimka("Nepodarilo se
zkopirovat indicie do statickeho pole a buffru.");
    if(!odemkniVrcholyProCteni()) throw StromVyjimka("nepodarilo se odemknout
vrcholy pro cteni.");
} catch (std::exception& e) {
    std::cout << "Model creation exception: " << e.what() << std::endl;
}
```

- fragment funkce generujVlastnostiVetvi(), v němž probíhá inicializace vlastností větví

```
/*
Klic slouzi k popisu zakladni vetvici struktury stromu
0x0000 znaci v klici vetev, ktera ma mit potomky, kteri se oba dale deli
0x0001 znaci v klici vetev, ktera se jiz nedeli
0x0002 znaci v klici misto, kde vetev neni
*/
for(int z = 0; z < druhStromu.urovenRozvetveni; z++) {      // v kazde rade
koruny
    n=pow(float(2), z);                                     // do n se ulozi,
kolik vetvi muze byt maximalne v teto rade
    for(int x = 0; x<n; x++) {                              // opakuj pro kazdou
z jednotlivych vetvi rady v klici
        if(klic[h::integratePower(2,z)+x] == 0x0000) {  // vetev ma mit
potomky, vygeneruj jejich vlastnosti
            vlastnostiVetvi[pocetV] =
generujVlastnostiVetve(vlastnostiVetvi[h::integratePower(2,z)+x - pocetVKT],
0, druhStromu);
            if(klic[h::integratePower(2,z+1)+2*x] == 0x0000) { // ma-li se
licha vetev dale delit, nastav k na false
                vlastnostiVetvi[pocetV].k = false;
            } else {
                vlastnostiVetvi[pocetV].k = true;           // licha vetev se
jiz dale nedeli, nastav k na true
            }
            pocetV++;       // inkrementuj citac jiz vygenerovanych vetvi
```

```
            vlastnostiVetvi[pocetV] =
generujVlastnostiVetve(vlastnostiVetvi[h::integratePower(2,z)+x - pocetVKT],
1, druhStromu);
            if(klic[h::integratePower(2,z+1)+2*x+1] == 0x0000) { // ma-li se
suda vetev dale delit, nastav k na false
                vlastnostiVetvi[pocetV].k = false;
            } else {                                          // suda vetev se
jiz dale nedeli, nastav k na true
                vlastnostiVetvi[pocetV].k = true;
            }
            pocetV++;
        } else if(klic[h::integratePower(2,z)+x] == 0x0002) {       // když
bude parent koncovou vetvi, pricti 1 do citace neexistujicich vetvi
            pocetVKT+=1;
        }
    }
}
```

- Použití třídy t::Strom3 v funkci se::Engine::pripravGeometrii() je skutečně jednoduché:

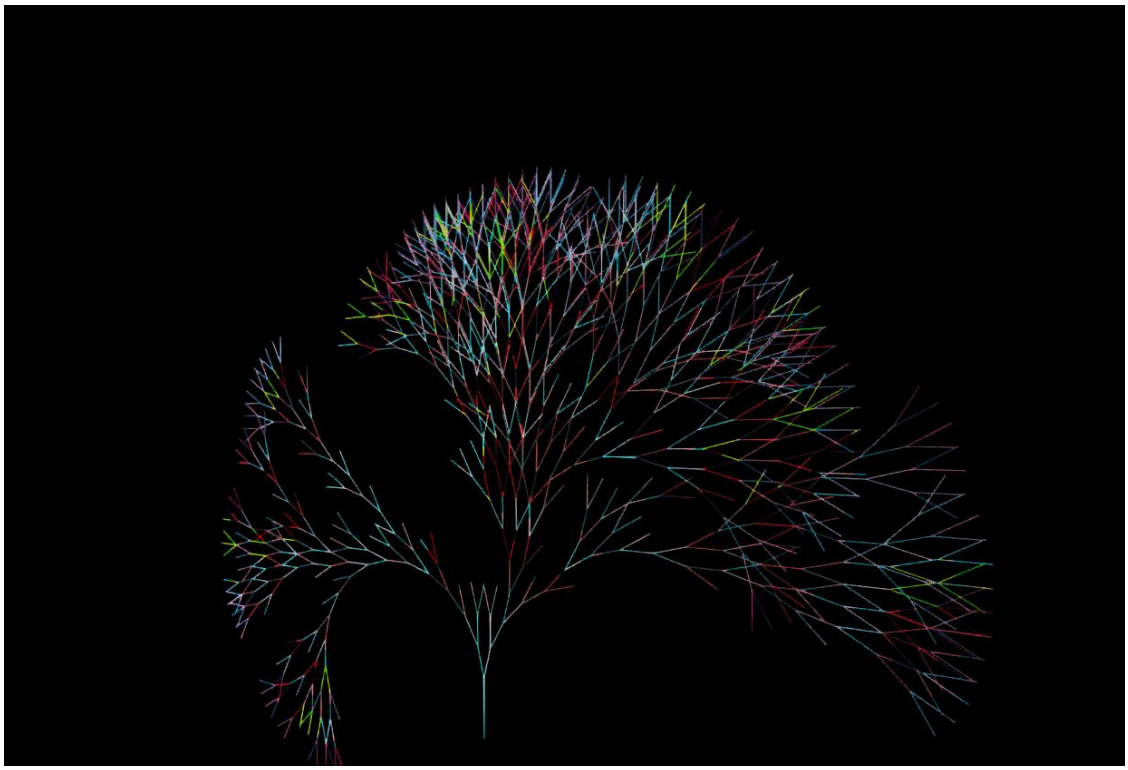
```
    t::DruhStromu druhStromu;       // pro popsani zakladnich parametru
stromu deklaruji instanci t::DruhStromu
    druhStromu.urovenRozvetveni = 10;       // pocet generaci vetvi
    druhStromu.pravdepodobnostRozvetveni = 85;      // pravdepodobnost kazde
vetve, jestli se rozvetvi
    druhStromu._iDType = 3;     // zpusob, jakym je vypoctena delka vetve
(dedeni delek vetvi)
    druhStromu._iRType = 4;     // zpusob, jakym je vypoctena rotace potomku
    druhStromu._iSType = 4;     // zpusob, jakym je vypocten sklon potomku
    druhStromu.rozliseniE = 5;  // kolika-boky hranol chceme
    druhStromu.rozliseniV = 11; // kolik hranolu maximalne muze mit vetev
    druhStromu.barva = t::cervena;      // barva materialu stromu

    D3DXMATRIX pocatek;                 // pozice stromu ve svete sceny
    fDalka = 1950000.f;                 // jak velka je dalekost dalky
    float xoffset = -2*fDalka;          // pozice x modelu
    float yoffset = -2*fDalka;          // pozice y modelu
    float zoffset = 0.f;                // pozice z modelu
    D3DXMatrixTranslation(&pocatek, xoffset, yoffset, 0.f);     // translace
matice pocatek
    stromy3D.emplace_back(druhStromu, pocatek, &pd3dZarizeni, 0.00f);     //
Vytvori strom a prida ho na konec naseho vektoru stromy3D
```

Ukázka z programu:

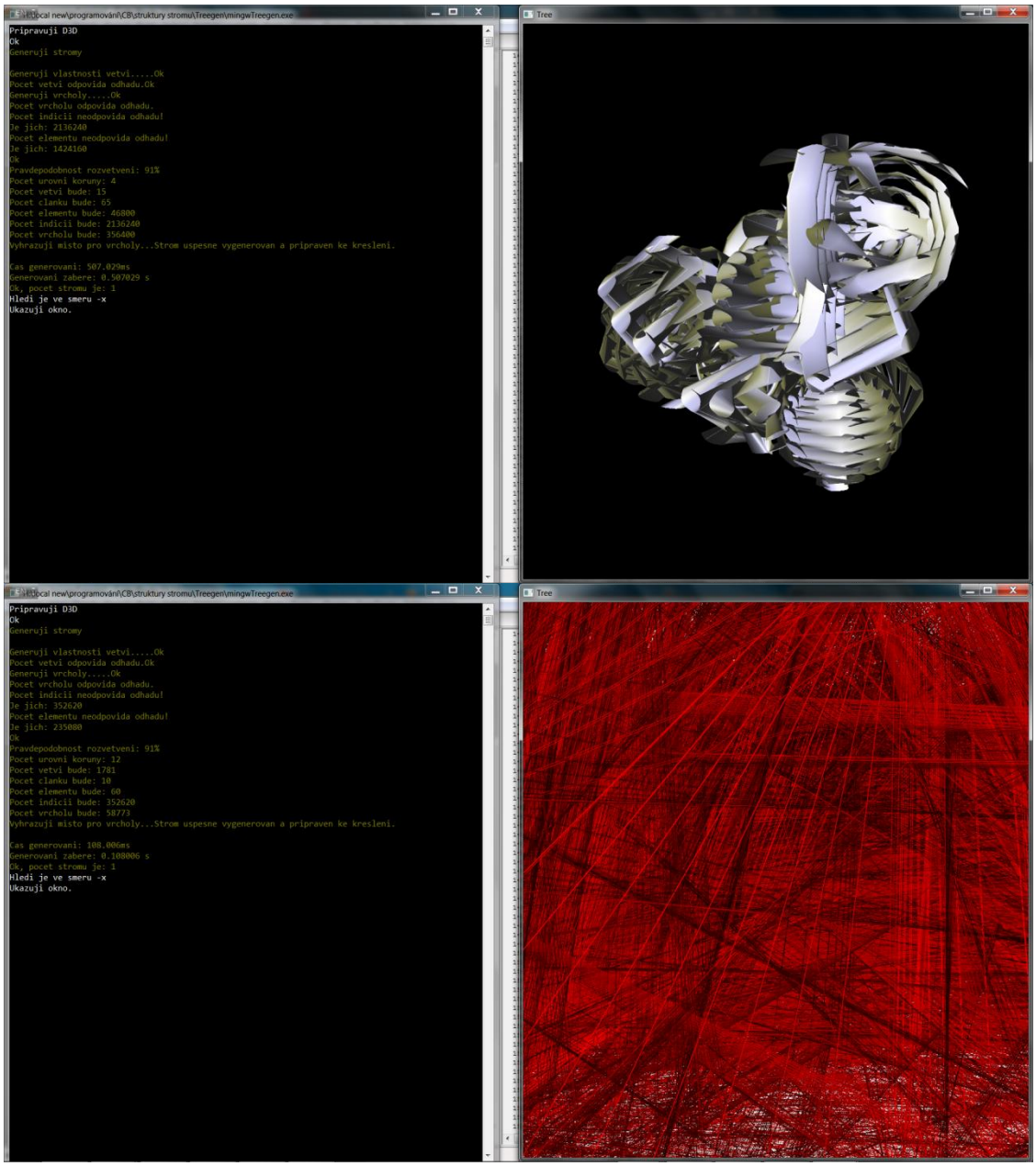
- Na závěr bych rád prezentoval ukázky z rozmanitých fází vývoje projektu:

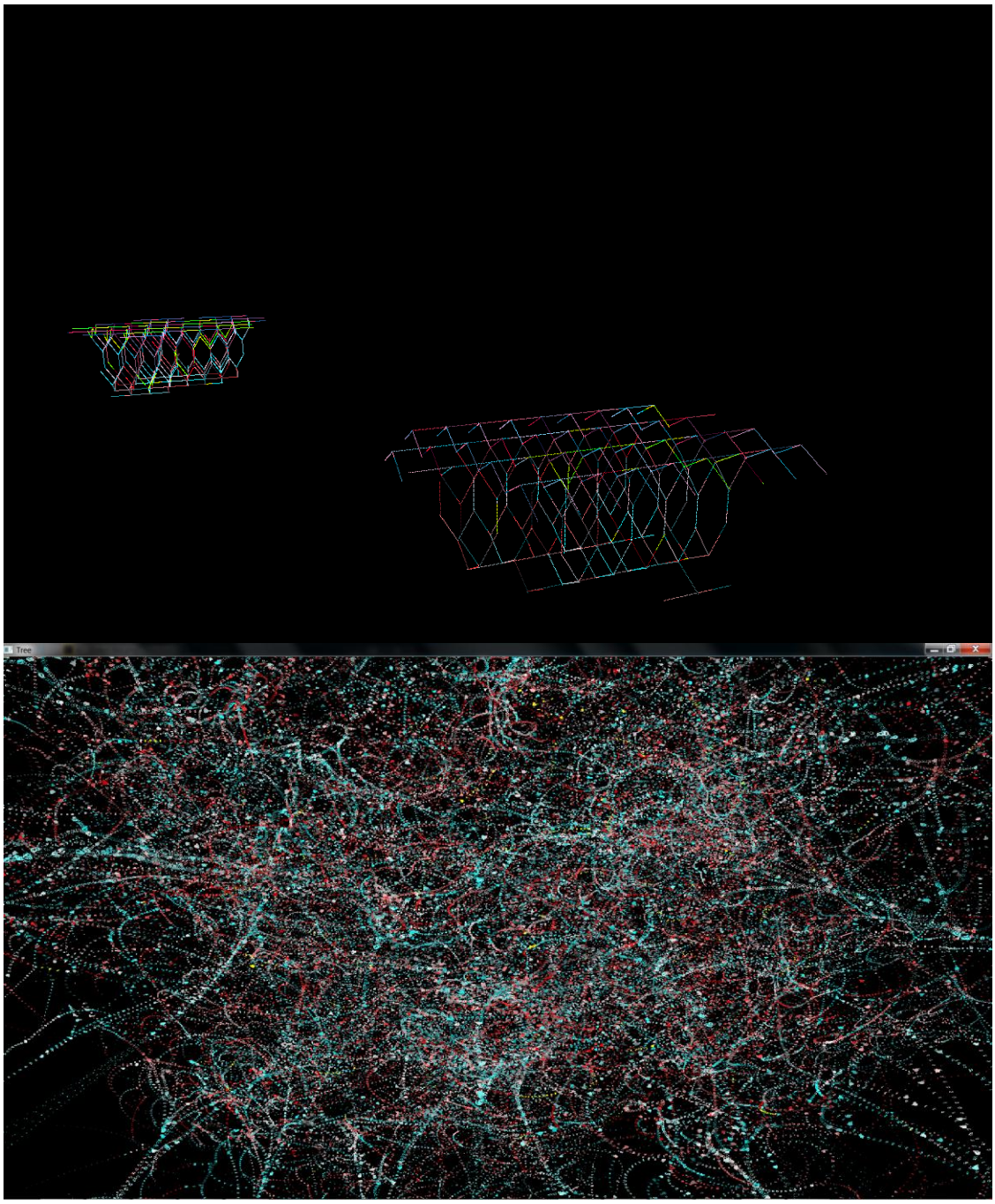
Na následujícím obrázku lze ukázat různé druhy struktur, které lze generovat. Část kódu generující strukturu lze upravovat s minimální znalostí programování a je dostatečně izolovaná od ostatního kódu.



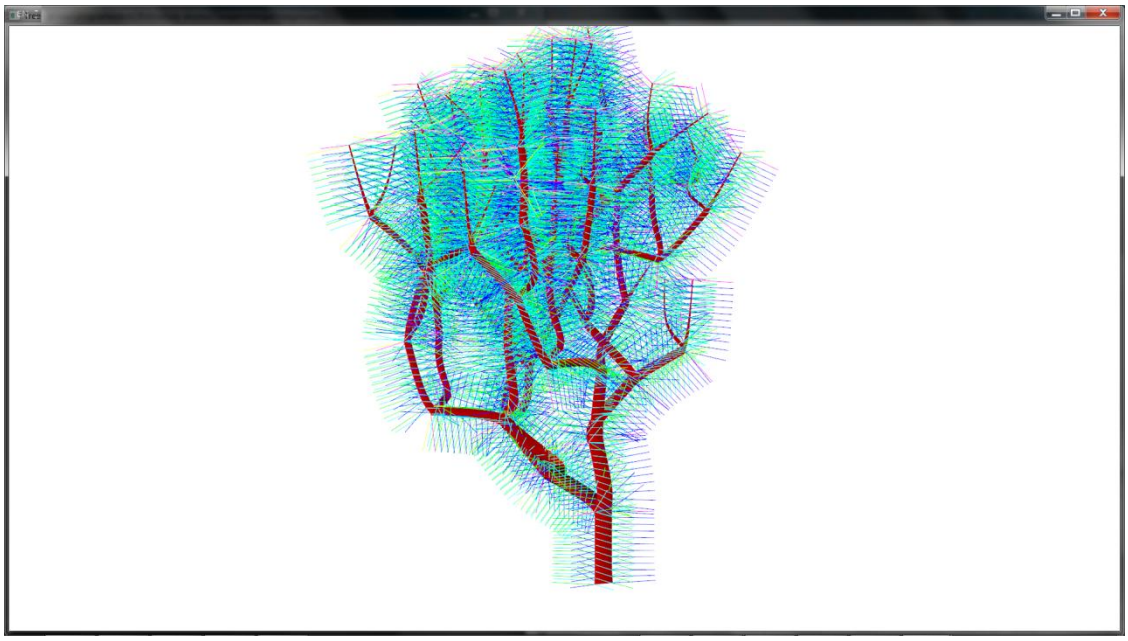
struktury

Na dalších obrázcích si lze všimnout multifunkčnosti programu, který lze použít i pro generování abstraktních modelů.


C:\Documents and Settings\Local new\programování\CB\struktury stromu\Treegen\mingwTreegen.exe
Pripravuji D3D
Ok
Generuji stromy
Generuji vlastnosti vetvi.....Ok
Pocet vetvi odpovida odhadu.Ok
Generuji vrcholy.....Ok
Pocet vrcholu odpovida odhadu.
Pocet indicii neodpovida odhadu!
Je jich: 2136240
Pocet elementu neodpovida odhadu!
Je jich: 1424160
Ok
Pravdepodobnost rozvetveni: 91%
Pocet urovni koruny: 4
Pocet vetvi bude: 15
Pocet clanku bude: 65
Pocet elementu bude: 46800
Pocet indicii bude: 2136240
Pocet vrcholu bude: 356400
vyhrazuji misto pro vrcholy...Strom uspesne vygenerovan a pripraven ke kresleni.
Cas generovani: 507.029ms
Generovani zabere: 0.507029 s
Ok, pocet stromu je: 1
Hledi je ve smeru -x
Ukazuji okno.
Tree
C:\Documents and Settings\Local new\programování\CB\struktury stromu\Treegen\mingwTreegen.exe
Pripravuji D3D
Ok
Generuji stromy
Generuji vlastnosti vetvi.....Ok
Pocet vetvi odpovida odhadu.Ok
Generuji vrcholy.....Ok
Pocet vrcholu odpovida odhadu.
Pocet indicii neodpovida odhadu!
Je jich: 352620
Pocet elementu neodpovida odhadu!
Je jich: 235080
Ok
Pravdepodobnost rozvetveni: 91%
Pocet urovni koruny: 12
Pocet vetvi bude: 1781
Pocet clanku bude: 10
Pocet elementu bude: 60
Pocet indicii bude: 352620
Pocet vrcholu bude: 58773
vyhrazuji misto pro vrcholy...Strom uspesne vygenerovan a pripraven ke kresleni.
Cas generovani: 108.006ms
Generovani zabere: 0.108006 s
Ok, pocet stromu je: 1
Hledi je ve smeru -x
Ukazuji okno.
Tree

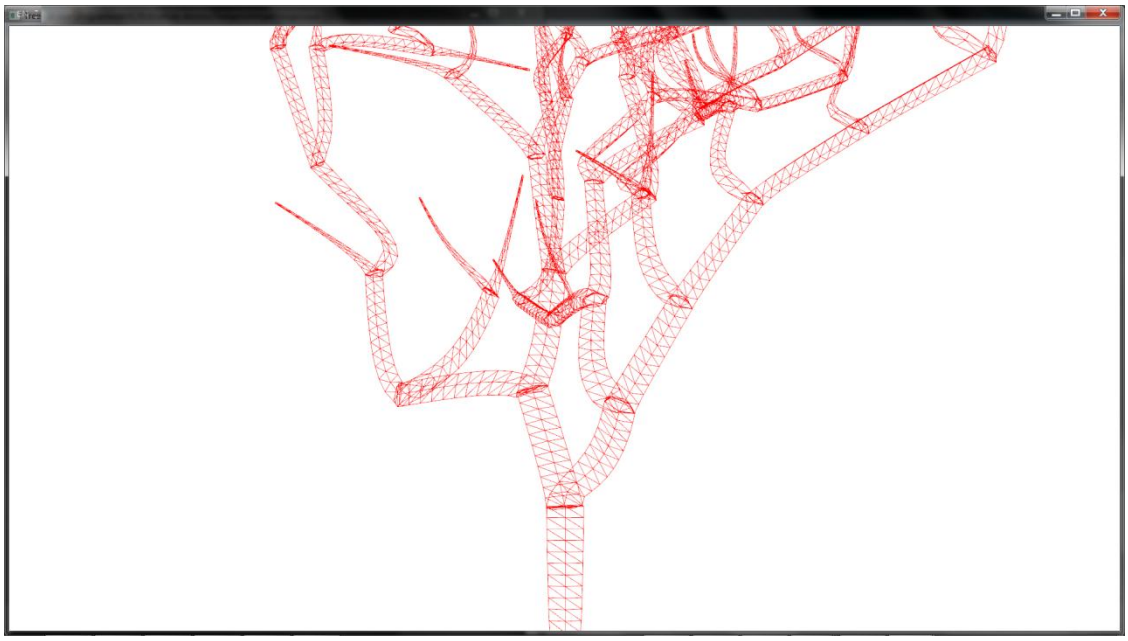


Tento obrázek ukazuje graficky znázorněné normály každého vertexu modelu. Grafickou reprezentaci normál je možné zapnout v souboru globals.h odkomentováním definice konstanty ZOBRAZ_NORMALY.



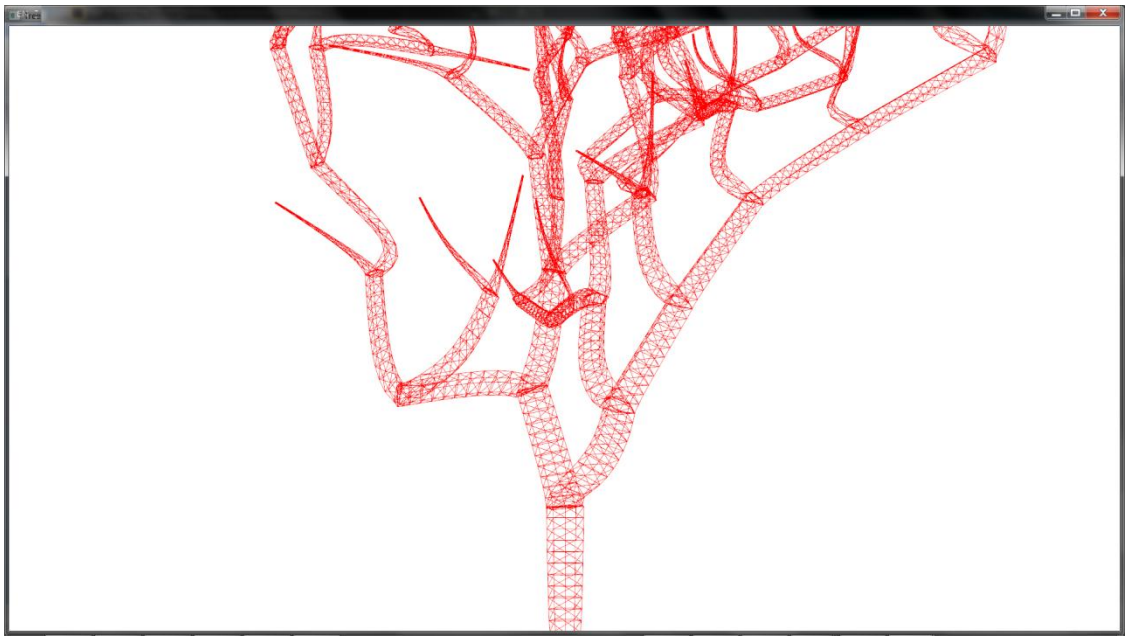
Každý vrchol musí mít svoji normálu, symbolika protikladů.

Na následující sérii obrázků lze demonstrovat interní použití indicií, které dávají grafické kartě informace o spojení vrcholů.



Rub není vidět.

Zde je patrná funkce cullingu, po vypnutí se vykreslují i skryté plošky z rubu modelu. Culling funguje na principu používání indicií jen v jednom směru, např. po směru hodinových ručiček.



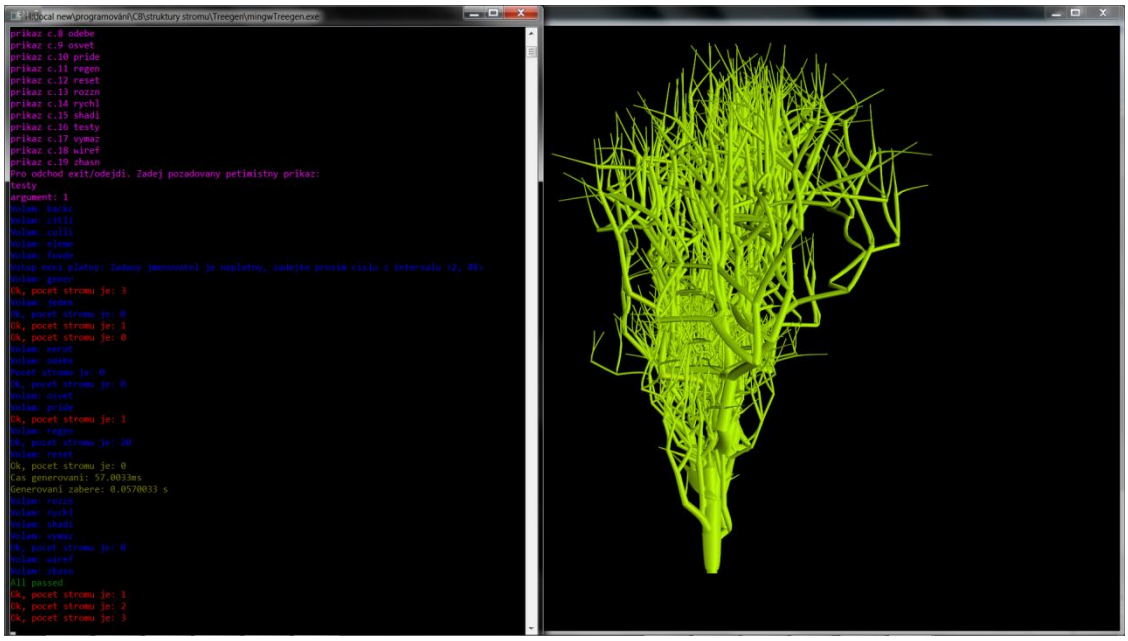
Je vidět i rub.

V konzoli je možné nechat si vypsat indicie pro snazší hledání indexovacích chyb. Toho lze dosáhnout odkomentováním definice konstanty VYTISKNI_INDICIE v souboru globals.h

Konzole a indicie

Následující dvojice obrázků ukazuje rozdíl mezi Goraud a Flat technikou stínování.

Následující obrázek ukazuje, jak všechny testy prošly.

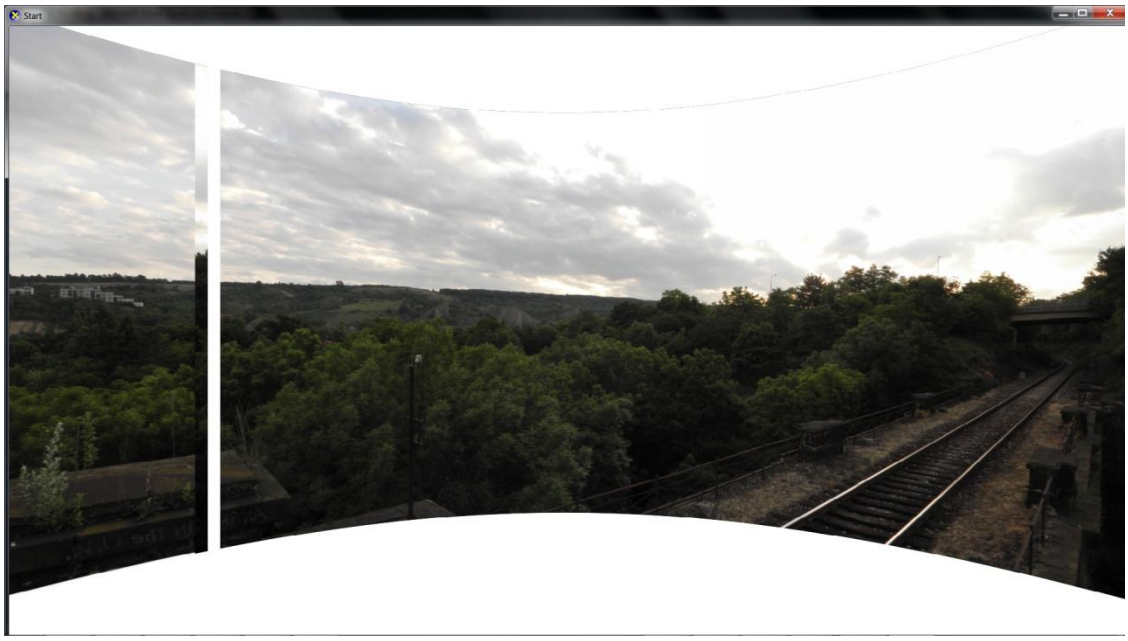


All passed!

Původní geometrický model měl obsahovat ještě přirozenější přechod dělících se větví, ten byl však zavrhnut pro přílišnou složitost indexování vrcholů.

Přechod

Na následujícím obrázku lze vidět panorama kolem Barrandovského mostu v Hlubočepích. Byla to snaha o přípravu texturování větví, kterou plánuji do budoucna.



Příroda!

Závěr

Myslím, že jsem napsal dobrý základ pro další vývin generátoru stromů. Věřím, že v budoucnu bude tímto programem možné generovat rozmanitou flóru. V blízké budoucnosti plánuji přidat textury listů, kůry a květů spolu s mipmapovými texturami, jež umožňují přidat textuře normálovou texturu a vytvořit tím iluzi trojrozměrnosti detailů textury.