Lavoro di maturità

Simulazione di frane e valanghe

Testo accompagnatorio con opera

Autore: Marco Ruggia
Relatore: Alberto Maraffio

Scuola cantonale grigione
2012

Indice

CD allegato:

- Il programma (versione ridotta)
- Il codice sorgente (senza DHM25 e SpotMosaic)
- Le foto del Cengalo (reale)
- I video di dimostrazione
- Alcuni link
- Questa relazione con le spiegazioni.

1 L'Obiettivo

L'obiettivo del lavoro di maturità è quello di creare un programma di computer in grado di simulare una frana o una valanga. È stata scelta la frana del Cengalo in Bregaglia scesa nel dicembre 2011 per rappresentare in modo esemplare tutte le altre.

2 Il programma

Il programma di simulazione è stato scritto partendo solo dal sistema operativo Windows7 e DirectX. Nel testo che segue, sono descritti i passi che sono stati necessari per crearlo e farlo funzionare. Tutte le parti di programma modificate o cancellate in fase d'elaborazione e quelle rimaste inutilizzate non sono riportate in questa presentazione.

Un accento è stato posto sulle formule di fisica e matematica che rappresentano il movimento dei corpi, per esempio anche della telecamera virtuale utilizzata. Per questo, sono spiegati esaustivamente solo i passi che hanno a che fare con la fisica e con la matematica. I problemi di programmazione sono stati volutamente tralasciati. A chi s'interessa al funzionamento del programma: cfr. il capitolo codice sorgente (ben commentato in inglese).

2.1 Le basi

La presentazione del progetto (e il programma stesso) è suddivisa in tre parti. La prima parte contiene i codici fondamentali del programma di simulazione.

2.1.1 Vedere il mondo virtuale

La prima cosa che ho dovuto fare è stata quella di trovare un modo per creare un mondo tridimensionale (3-D) e poterci "guardare dentro" attraverso lo schermo. Cercando in internet ho scoperto che per questo proposito esiste già un'API (Application Programming Interface) molto popolare di nome DirectX che fa esattamente quanto mi serve. Fondamentalmente, questa API è come una telecamera. Bisogna dirle dove si trova, da che parte sta guardando e quanto zoom ha l'obiettivo. In cambio l'API crea l'immagine di che cosa vede. Sfortunatamente, l'API é avviabile solo per la lingua di programmazione C/C++ con la quale non avevo esperienza. Quindi ero confrontato con frasi del tipo:

> "To create an instance of the device we use our main Direct3D object pointer which provides the following function (the *definition* is shown):
>
> HRESULT IDirect3D9::CreateDevice(UINT adapter, D3DDEVTYPE deviceType …"
> (secondo punto dopo l'introduzione)

Nessuna idea di che cosa volessero dire. Dopo circa un mese ho superato l'ostacolo; non spiego in dettaglio che cosa ho dovuto studiare perché è molto complicato, lungo e anche un po' noioso da esporre.

2.1.2 Muoversi nel mondo virtuale

Con il programma com'era in quel momento, l'unico modo per muoversi era cambiare le coordinate della "telecamera" manualmente, cioè spegnere il programma, cambiare qualche numero e poi riaccenderlo. Quindi la prossima cosa da fare era dare all'utente la possibilità di muoversi in questo mondo con la tastiera e il mouse senza bisogno di uscire dal programma.

Qui di seguito spiego come ho fatto: Come già detto, le caratteristiche della "telecamera" sono definite dalla sua posizione$A$, dal punto in cui guarda$B$ e dallo zoom (che però, nel caso del mio programma è fisso).Inoltre, c'è la variabile v che definisce la sua velocità di movimento. La variabile v è inversamente proporzionale al no. di fotogrammi per secondo (fps), cosi ché la velocità di movimento rimanga costante.

Per ogni fotogramma il computer fa questi calcoli:

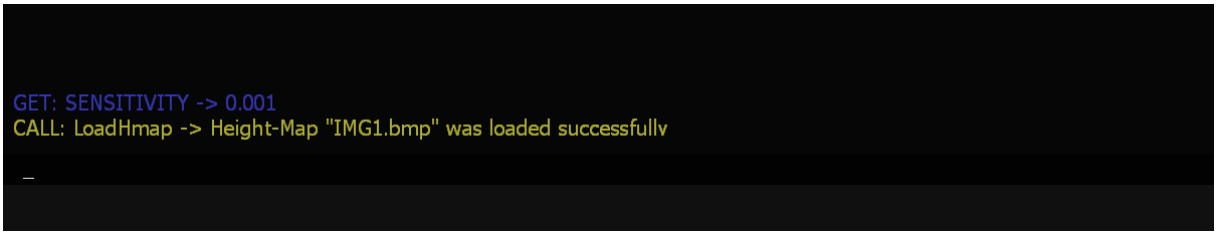
- Muoversi in avanti/dietro: $A = A \pm \overline{AB} * v \quad B = B \pm \overline{AB} * v$
- Muoversi a sinistra/destra: $A = A \pm \overline{AB} \times \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} * v \quad B = B \pm \overline{AB} \times \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} * v$
- Girarsi a sinistra/destra: $B = R * \overline{AB} * (\pm v) + A$

  R: matrice di rotazione sull'asse$\left(\overline{AB} \times \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \right) \times \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$
- Girarsi in su/giù: $B = R * \overline{AB} * (\mp v) + A$

  R: matrice di rotazione sull'asse$\overline{AB} \times \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$

2.1.3 La console

Dall'inizio del progetto erano ormai passate sei settimane e lo schermo, pur sapendo come far girare la telecamera, era ancora nero. Per sapere se quello che facevo funzionava mi serviva un modo per richiedere informazioni al programma (per esempio: questo ordine ha funzionato? oppure, dove si trova la telecamera, risp. dove mi trovo?). Così ho pensato di fare una console di comando e metterla al bordo dello schermo. Questa console doveva saper fare tre cose: permettermi di scrivere dei comandi, capire ed eseguire i comandi ricevuti e mostrarmi la risposta (per esempio: ha funzionato / non ha funzionato, oppure no. fps). Ecco il risultato:

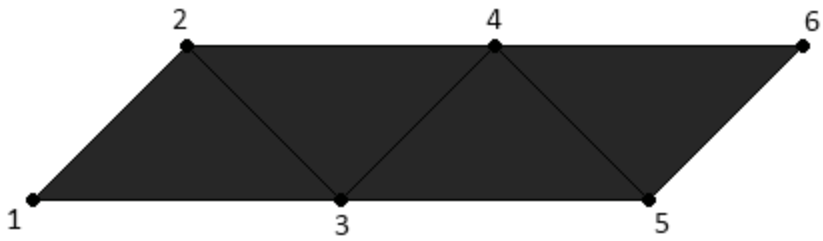


2.2 Il Cengalo

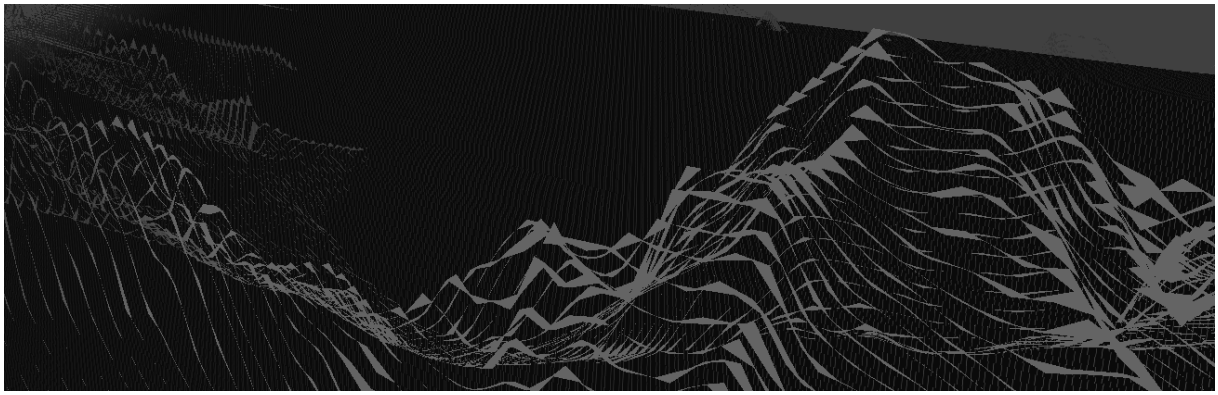
Nella seconda parte del progetto l'obiettivo era visualizzare il Cengalo. Grazie alla Scuola cantonale ho ricevuto il modello altitudinale DHM25 e le immagini satellitari Spot Mosaic dell'ufficio federale di topografia swisstopo. Il CD del DHM25 era pieno zeppo di tre bilioni e mezzo di righe come queste:

```
639375.000134000.000  2474.123
639400.000134000.000  2455.990
639425.000134000.000  2439.982
639450.000134000.000  2423.103
```

Ogni riga era composta da latitudine, longitudine e altitudine. Avendo però già scelto di usare DirectX ho dovuto seguire le sue regole. Una è il formato che devono avere gli oggetti nel mondo virtuale. Essi sono composti da una lista di coordinate (x, y, z) che DirectX interpreta nel seguente modo:



Però sul CD swisstopo le coordinate erano scritte riga per riga (1, 3, 5, …), quindi se avessi passato a DirectX la lista delle coordinate DHM25 così com'erano, il risultato sarebbe stato qualcosa del genere:



In altre parole, la sequenza delle coordinate è strutturata in modo diverso tra DirectX e swisstopo. Per evitare questa incongruenza ho dovuto creare una nuova lista coerente con l'ordine d'interpretazione DirectX.

Questa è la funzione che calcola la riga nel DHM25 con le coordinate x e y del mondo virtuale:

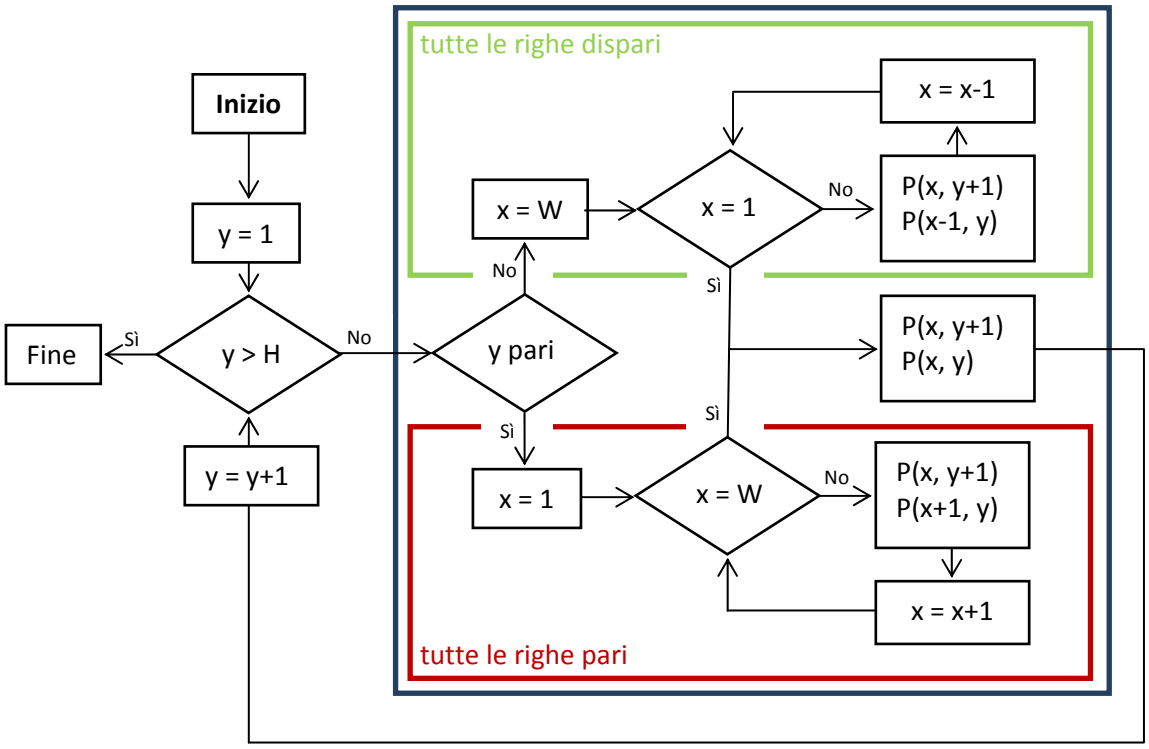
$$P(x,y) = (y-1) * H + x - 1 \qquad (-1\ perchè\ il\ PC\ comincia\ a\ contare\ da\ 0)$$

$$H: Lunghezza\ del\ mondo$$

$$W: Larghezza\ del\ mondo$$

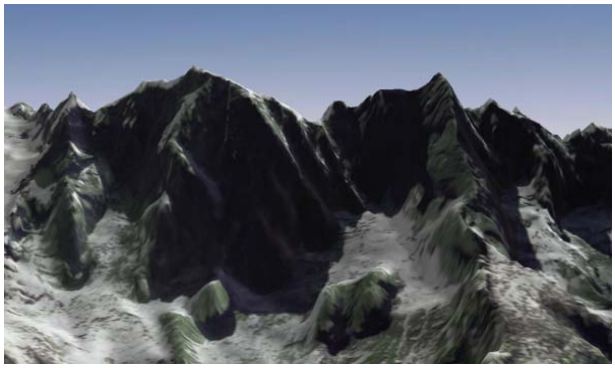
Ogni volta che viene chiamata, la riga DHM25 viene calcolata e scritta alla fine della nuova lista che riceverà DirectX.

Questo diagramma descrive secondo quale principio viene creata la lista:



Detto in parole semplici il programma passa le righe a zig-zag, lo fa sempre una volta da destra a sinistra e poi una da sinistra a destra finche non ci sono più righe.

Aggiungere l'immagine satellitare non è stato un problema perché le coordinate combaciavano con quelle della lista e DirectX lo interpretava correttamente. Questo è il risultato generato dal PC in confronto all'originale (scoprite le differenze e lamentatevi da swisstopo):



2.3 La fisica

La terza e ultima parte é stata allo stesso tempo anche la più interessante. La parte fisica tratta l'aspetto della simulazione, che è quello per il quale ho scelto questo lavoro.

2.3.1 Movimento

Quando un sasso o qualsiasi altro corpo si muove nell'aria ha una posizione e una velocità variabili che dipendono da certi fattori. Il movimento è rappresentabile con queste due formule. Con ciò può essere anche calcolato, cioè sapendo da dove parte l'oggetto, prevedere dove arriva.

$$\vec{v} = \overrightarrow{v_0} + \vec{g} * \Delta t \qquad \vec{r} = \overrightarrow{r_0} + \vec{v} * \Delta t$$

$\vec{v}, \overrightarrow{v_0}$: *velocità del sasso nuova e precedente*
$\vec{r}, \overrightarrow{r_0}$: *vettore posizione nuovo e precedente*
Δt: *tempo passato dall'ultimo calcolo*

2.3.2 Determinazione dell'altitudine

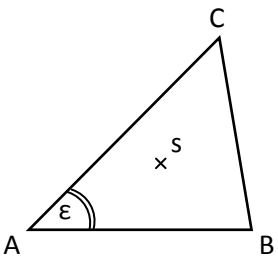
Prima o poi, ogni sasso che precipita tocca terra. Dove cade esattamente? Laddove le coordinate tridimensionali del sasso corrispondono a quelle del terreno.
Il prossimo calcolo serve a determinare per ogni fotogramma, cioè per ogni unità di tempo, l'altitudine del terreno sotto il sasso. Il programma calcola una serie di posizioni che si succedono nel tempo (serie di fotogrammi) in base a una certa regola di movimento (quella del capoverso precedente). Rilevante è sapere in quale fotogramma il sasso tocca il triangolo che si trova sulla sua verticale.
In ogni momento in cui è calcolata, la posizione orizzontale del sasso corrisponde a quella del preciso triangolo che si trova sotto la sua verticale: i parametri x e y corrispondono sempre (in qualsiasi luogo si trovi il sasso, sotto c'è uno dei triangoli che formano il mondo virtuale). La successione dei calcoli serve a riconoscere quando l'altitudine del sasso corrisponde a quella del punto sul piano inclinato formato dal triangolo del terreno che contiene la sua stessa posizione (coordinate e altezza). Quando il parametro z corrisponde, il sasso ha toccato terra.
Ricercata è la coordinata z del punto s (sasso).

$$\vec{s} = \overrightarrow{AB} * u + \overrightarrow{AC} * v$$

$$\vec{s}_x = \overrightarrow{AB}_x * u + \overrightarrow{AC}_x * v \qquad \vec{s}_y = \overrightarrow{AB}_y * u + \overrightarrow{AC}_y * v$$

$$\downarrow$$

$$u = \frac{\overrightarrow{AC}_y * s_x - \overrightarrow{AC}_x * s_y}{\overrightarrow{AC}_x * \overrightarrow{AB}_y - \overrightarrow{AC}_y * \overrightarrow{AB}_x} \quad v = \frac{\overrightarrow{AB}_y * s_x - \overrightarrow{AB}_x * s_y}{\overrightarrow{AC}_x * \overrightarrow{AB}_y - \overrightarrow{AC}_y * \overrightarrow{AB}_x}$$



$$\boldsymbol{s_z = \overrightarrow{AB}_z * u + \overrightarrow{AC}_z * v}$$

2.3.3 Impatto col terreno

Quando il sasso tocca per terra rimbalza di una certa misura. Quanto, viene calcolato così (ricercata è la velocità dopo l'impatto):

$$\vec{N} = \overline{AB} \times \overline{AC}$$

$$\vec{v} * \vec{N} = |\overrightarrow{v_n}| * |\vec{N}| \rightarrow |\overrightarrow{v_n}| = \frac{\vec{v}*\vec{N}}{|\vec{N}|}$$

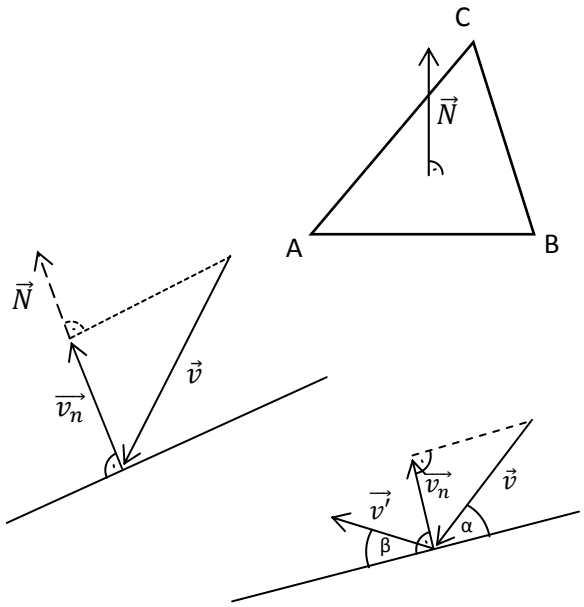
$$|\vec{N}| * x = |\overrightarrow{v_n}| \rightarrow x = \frac{|\overrightarrow{v_n}|}{|\vec{N}|}$$

$$\overrightarrow{v_n} = \vec{N} * x$$

$$\overrightarrow{v_n} = \frac{\vec{v}*\vec{N}}{|\vec{N}|^2} * \vec{N} \qquad (\overrightarrow{v_p} = \vec{v} + \overrightarrow{v_n})$$

$$\overrightarrow{\boldsymbol{v'}} = (\boldsymbol{k} + \boldsymbol{1}) * \overrightarrow{\boldsymbol{v_n}} + \overrightarrow{\boldsymbol{v}}$$

(k: coefficientedi restituzione)



Come risulta dalla formula, il coefficiente di restituzione è variabile, cioè a seconda del tipo di terreno e dell'oggetto più o meno compatti e elastici se ne può scegliere uno più alto – il sasso rimbalza come una palla – o più basso – il sasso si ferma già al primo impatto -. Per la mia simulazione ho scelto un coefficiente dell'1.5% (in un ipotetico piano orizzontale, un rimbalzo di 100 m è ridotto a 1,5 m).

2.3.4 Rotazione

Per l'impatto dei corpi sul terreno bisogna considerare anche la rotazione. A seconda di come ruotano e della forma che hanno, rispettivamente di che parte presentano all'impatto, i sassi rimbalzano in modo diverso. Questo aspetto l'ho risolto assieme al rimbalzo in un'altra formula molto bella che pero non viene usata nel programma perché non compatibile con le altre. Per compensare la rotazione mancante basta scegliere un coefficiente di restituzione più basso.

2.3.5 Attrito col terreno

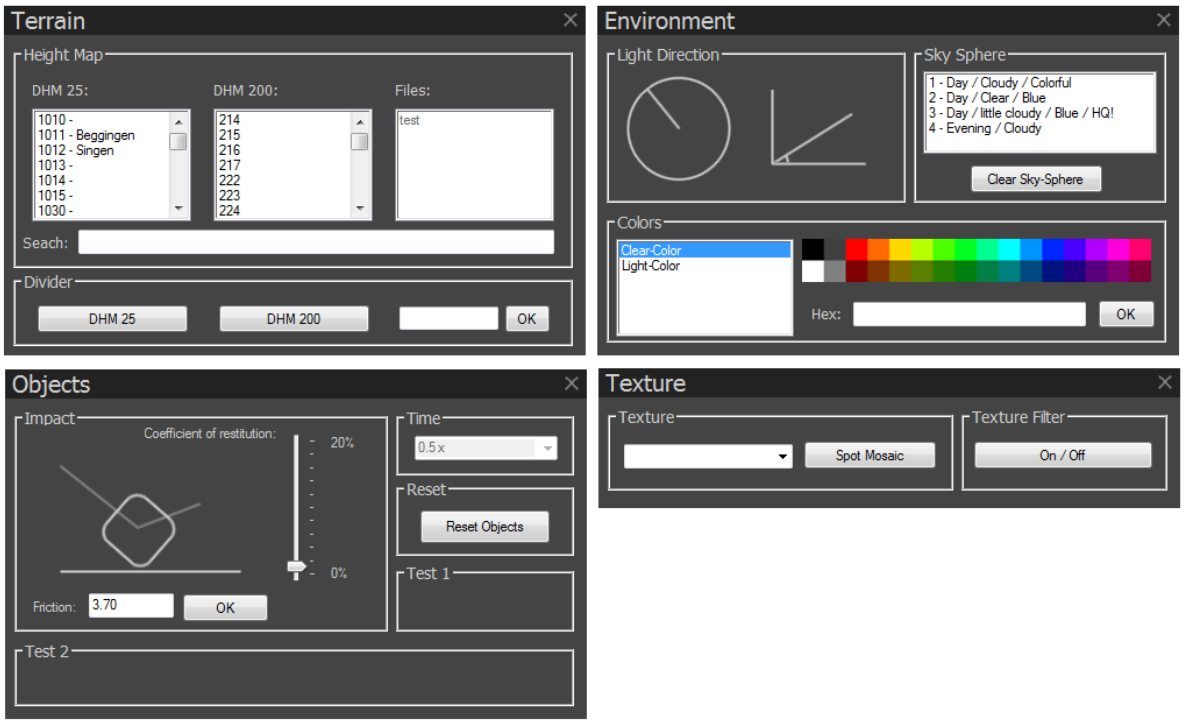
A un certo punto il sasso non avrà più energia per rimbalzare e prima di fermarsi inizierà a scivolare sul terreno. Scivolerà in questo modo (ricercata è la velocità di scivolamento nell'intervallo di tempo dato):

$$\overrightarrow{F_r} = -\frac{\overrightarrow{v_p}}{|\overrightarrow{v_p}|} * |\overrightarrow{F_n}| * \mu \qquad (\overrightarrow{F_n},\ \overrightarrow{F_p} \text{ come } \overrightarrow{v_n}, \overrightarrow{v_p})$$

$$\overrightarrow{\boldsymbol{v'}} = \overrightarrow{\boldsymbol{v_p}} + \frac{(\overrightarrow{\boldsymbol{F_p}}+\overrightarrow{\boldsymbol{F_r}})}{\boldsymbol{m}*\Delta\boldsymbol{t}}$$

2.4 Strumenti di modifica

Gli strumenti di modifica hanno un intento pratico perché liberano dalle scomodità d'uso della console. Di per se non sono necessari al buon funzionamento del programma. Nel programma essi sono divisi in quattro finestre che si possono aprire, chiudere e muovere a piacimento.



Dato che C++ è una lingua di programmazione complicata e perché dopo tutti i tutorial di DirectX già faticosamente letti non avevo più voglia di leggerne altri, ho programmato queste finestre in un linguaggio poco professionale, ma a me ben noto di nome AutoIT con il quale è abbastanza semplice operare.

In queste finestre ho poi aggiunto tante altre funzioni più o meno utili che mi hanno dato spunto anche per estendere la funzionalità della console. Qui sotto una piccola lista di parametri che si possono cambiare:

- L'immagine satellitare
- Il tempo che fa
- La cartina caricata
- La velocità della telecamera
- Il colore della luce
- La direzione della luce
- Il coefficiente di restituzione
- Il coefficiente d'attrito
- E così via

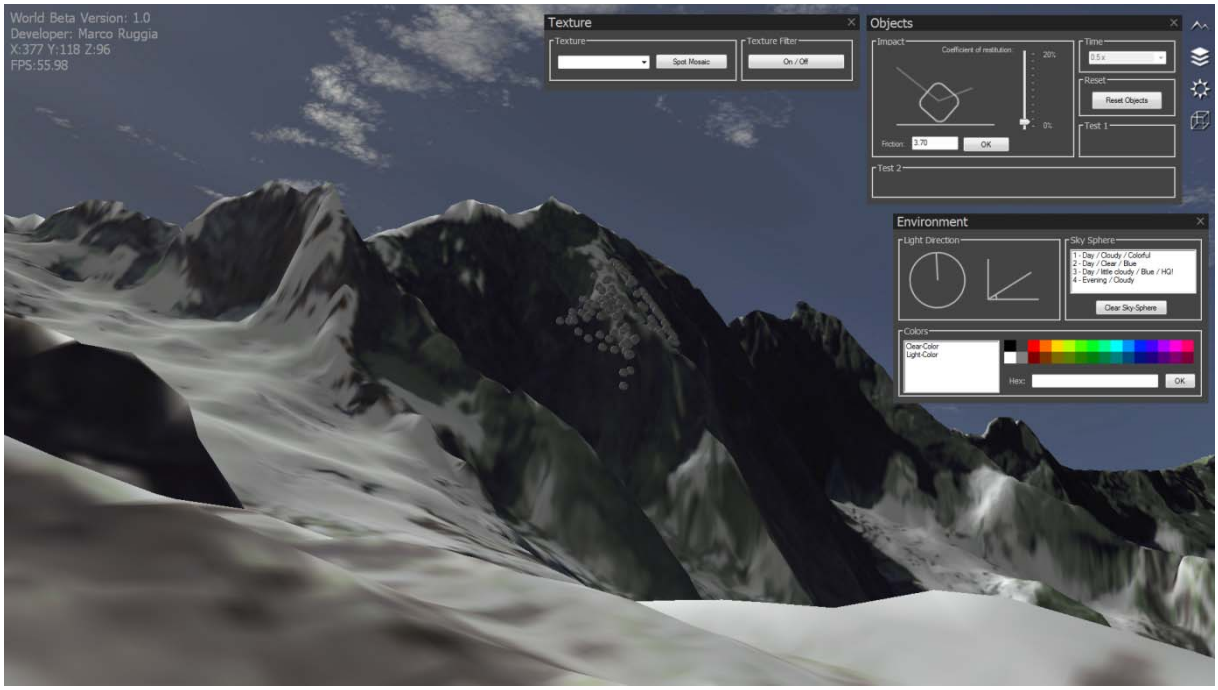
Il programma permette di modificare a piacimento la forma, la quantità e la posizione di partenza dei sassi. Si può così far scendere una frana di sassi a forma di cubo, sfera, cilindro o quant'altro si voglia da qualsiasi montagna compresa nel mondo virtuale, cioè di tutta la svizzera.

Per fare una valanga si diminuisce la dimensione dei corpi, si aumenta la loro quantità e superficie di partenza e si diminuisce l'attrito.

3 Conclusione

Tutti gli obiettivi sono stati raggiunti: La frana simulata corrisponde a quella reale e il programma é pronto per simularne altre.

Fare questo programma è stato però un lavoro molto lungo e difficile. Credo comunque che ne sia valsa la pena. Provatelo, e se avete qualche difficoltà a farlo partire, non sono reperibile.



Dichiarazione

Il sottoscritto Marco Ruggia dichiara di aver compilato e redatto di persona il lavoro di maturità. Dichiara pure di non aver commesso plagio e di aver indicato chiaramente e coscienziosamente le parti prese dalle diverse fonti.

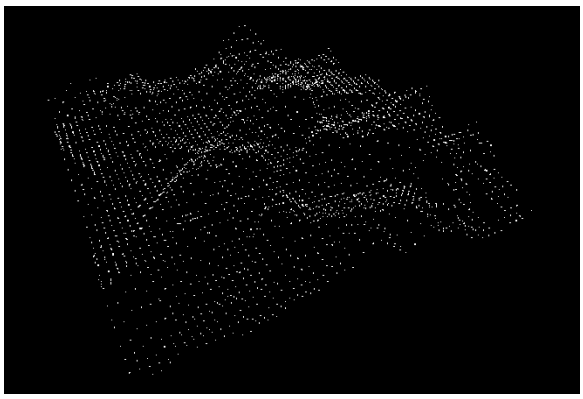
Luogo, data e firma

……

4 Diario di lavoro

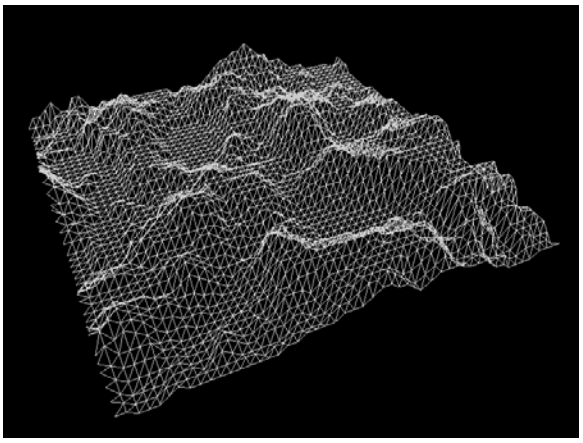
4.9.2011
v: 0.1

- *Creato progetto*
- *DirectX9 SDK installato*
- *Finestra*
- *Events handling*
- *DirectX 9 installato*



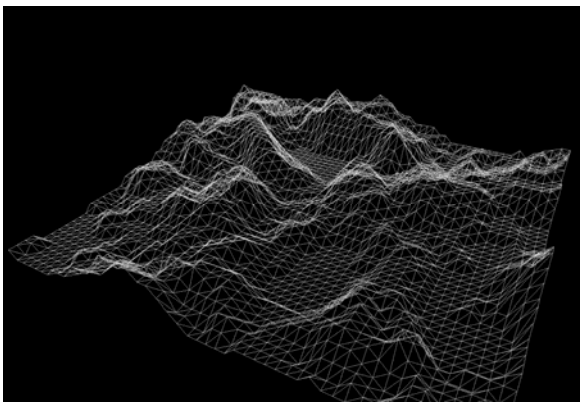
5.9.2011
v: 0.11

- *Height-Map class*
- *Leggere file BMP*
- *Convertire dati BMP in vertex*
- *Disegnare vertici*



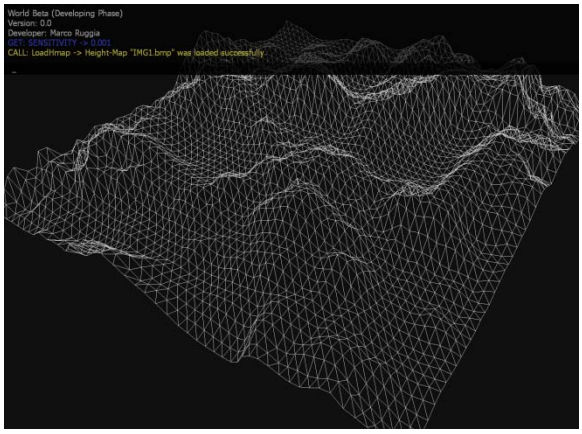
12.9.2011
v: 0.12

- *Modalità di disegno cambiata in linea*
- *Cambiato ordine dei vertex per creare rete*



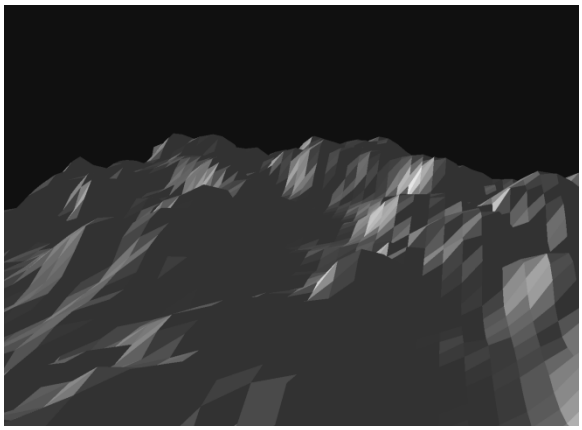
15.9.2011
v: 0.13

- *DirectInput 8*
- *Telecamera muovibile*
- *Opzioni per movimento veloce e lento*



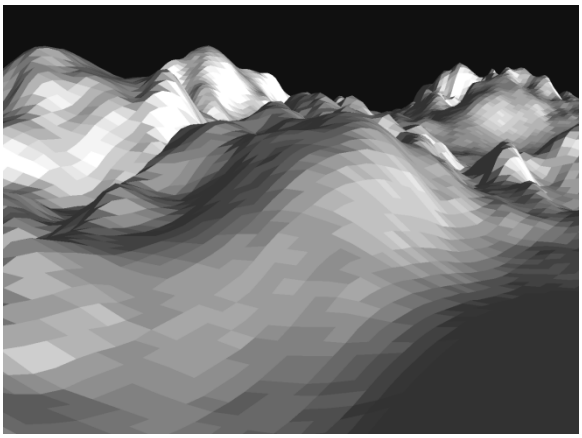
17.9.2011
v: 0.14

- *Console*
- *Riconoscimento di comandi*
- *Comandi per cambiare parametri e chiamare funzioni*



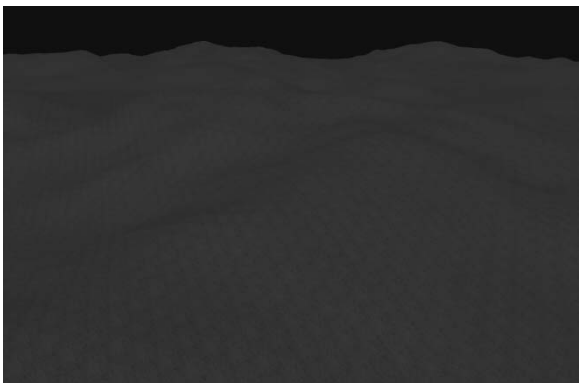
12.10.2011
v: 0.15

- *Triangoli "colorati" ->adeguamento del ordine dei vertex*
- *Vettori normali sui triangoli*
- *Attraverso normali calcolare luce*



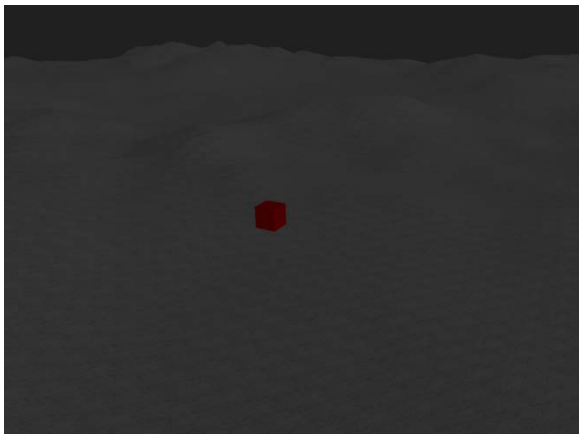
13.10.2011
v: 0.16

- *Vertex divisi in tanti buffer -> carte più grandi, migliore qualità*



15.10.2011
v: 0.17

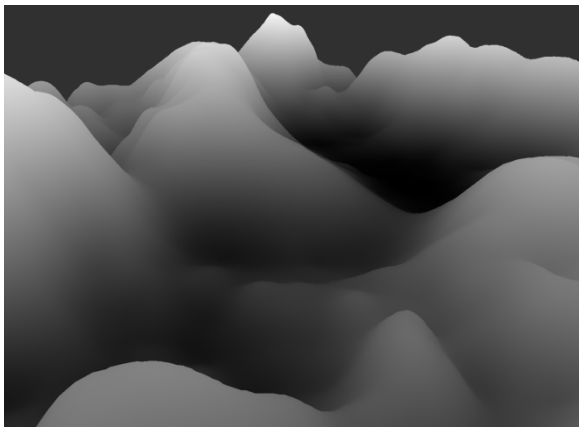
- *Coordinate texturiali aggiunte*
- *Texture*



27.10.2011
v: 0.2

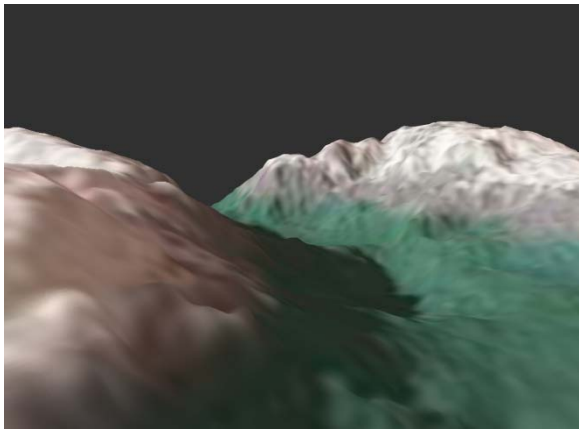
- *Oggetti*
- *Calcolo dei fotogrammi per secondo*
- *Forze di Gravità e vento*
- *Fisica durante l'impatto*

Video 1 & 2



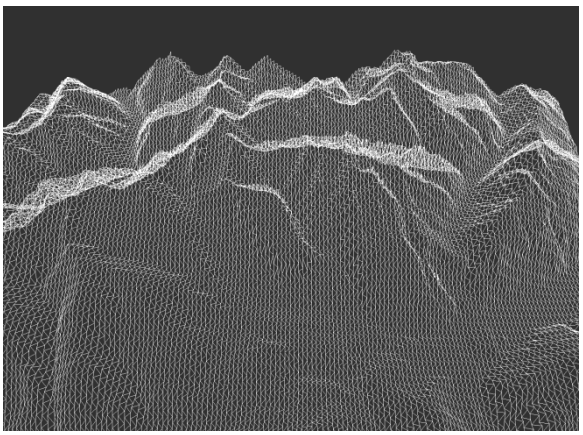
13.12.2011
v: 0.3

- *Combinazione vertex, index -> molto più veloce*
- *Oggetti cancellati perché non più compatibili*



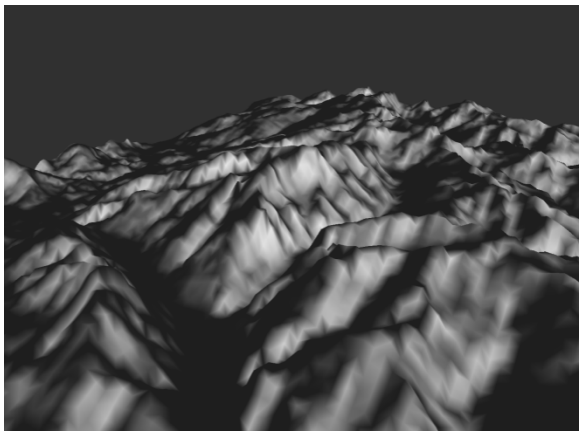
20.12.2011
v: 0.31

- *Cancellato coordinate texturiali, aggiunto colori per i vertex*



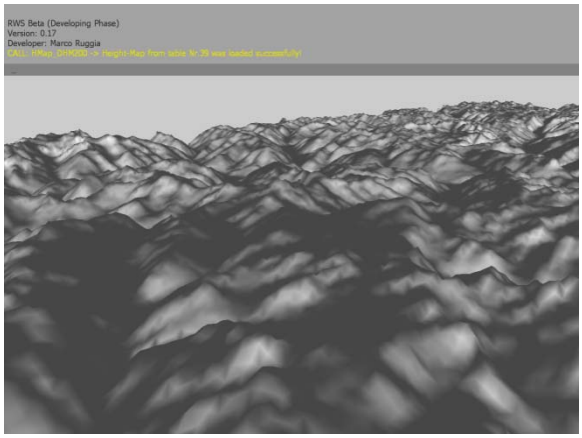
5.1.2012
v: 0.3

- *Scaricato dati DHM200*
- *Creato server SQL locale*
- *Dati DHM200 copiati in tabella SQL*
- *SQLAPI aggiunta*
- *Caricato cartina DHM200 attraverso SQL*



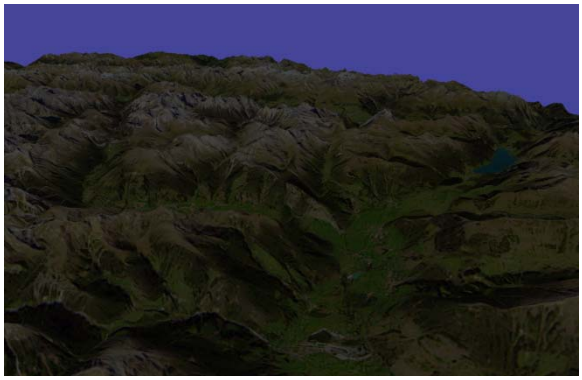
6.1.2012
v: 0.31

- *Dati DHM200 spostati su server esterno (accesso dappertutto)*
- *transizione della luce senza soluzione di continuità*



8.1.2012
v: 0.32

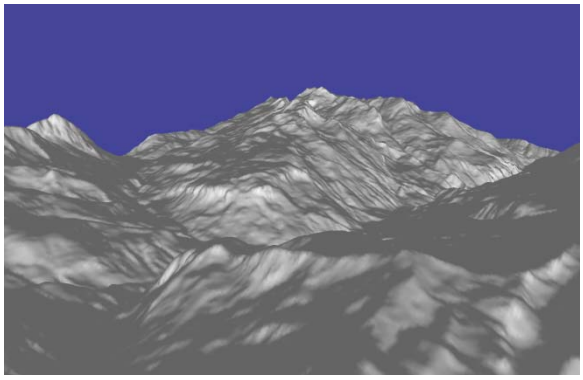
- *Tabella SQL divisa in segmenti. Facilita caricare una regione*
- *Aggiunto comando DHM200 per la console*



21.1.2012
v: 0.33

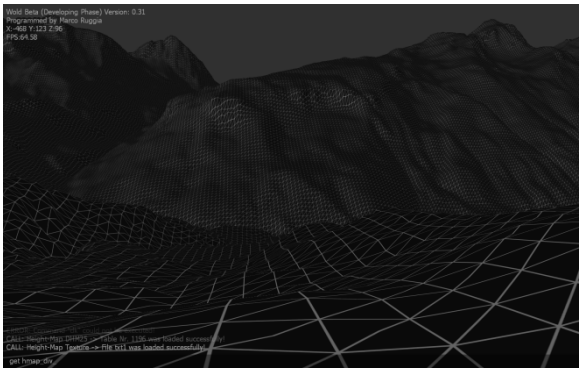
- *Immagine satellitare(Landsat Mosaic)*
- *Textura*

Video 3



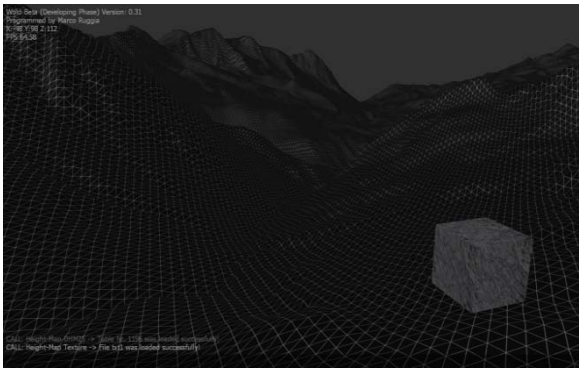
23.2.2012
v: 0.34

- *DHM25*



28.2.2012
v: 0.35

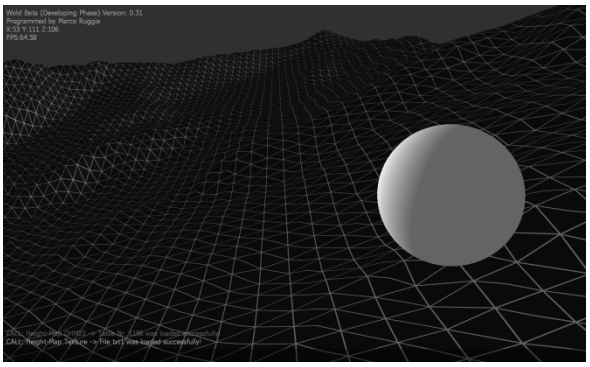
- *Nuova GUI*
- *Dati DHM spostati da SQL a file locali*
- *Texturacambiabile attraverso console*



14.3.2012
v: 0.4

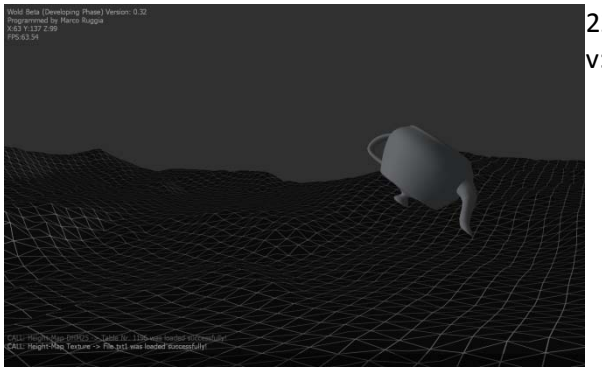
- *Raggiunto Oggetti*
- *Determinazione dell'altitudine più precisa*
- *Impatto col terreno*

Video 4



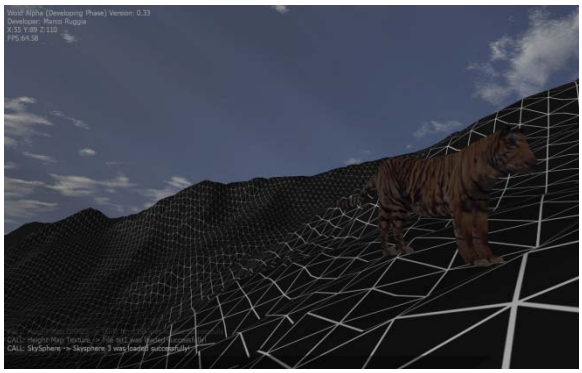
17.3.2012
v: 0.41

- *Mesh*
- *Leggere vertex dal mesh e ordinarli*
- *World Transform con matrice*
- *Forma dell'oggetto inclusa nella determinazione dell'altitudine*
- *Coefficiente di restituzione*



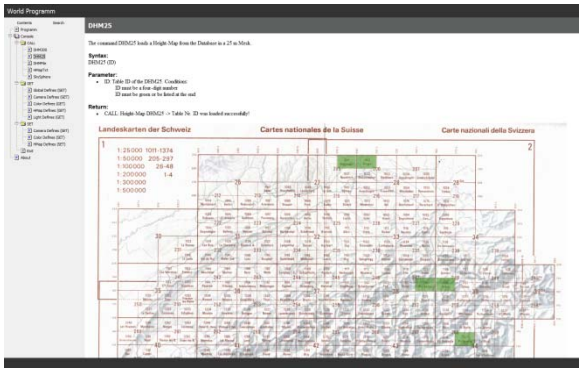
22.3.2012
v: 0.42

- *Rotazione con velocità costante*
- *Considerazione della rotazione durante l'impatto*
- *Cambiato sistema di coordinate. Più compatibilità con DirectX*



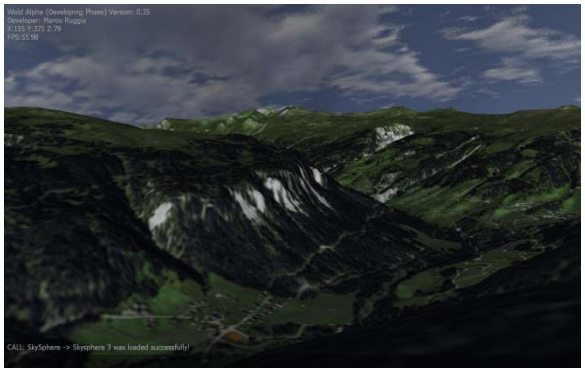
23.3.2012
v: 0.43

- *Caricare Oggetti da file .x*
- *Oggetti con texture o colori*
- *Aggiunto cielo*
- *Comando per cambiare il tempo*



28.3.2012
v: 0.5

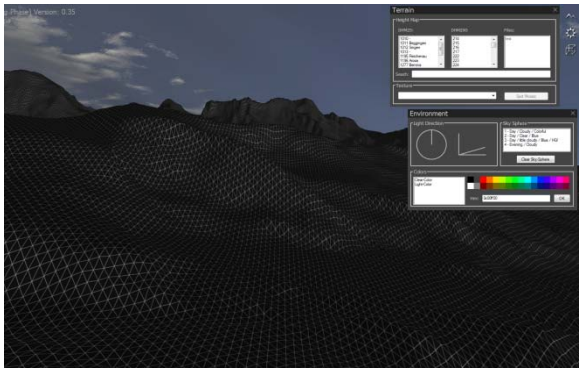
- *Documentazione dettagliata scritta*
- *Funzione di ricerca per la documentazione*



6.4.2012
v: 0.51

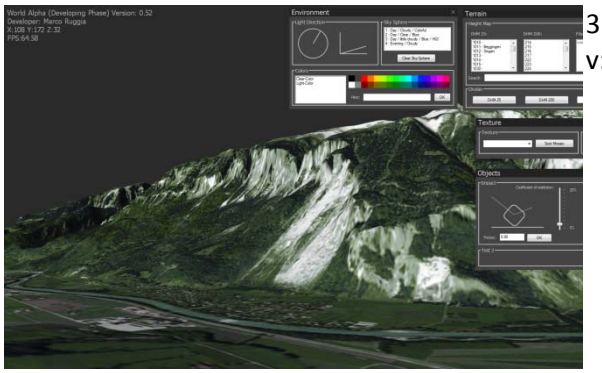
- *Immagini satellitari Spot-Mosaic*
- *Qualsiasi numero di oggetti contemporaneamente*

Video 5,6



25.4.2012
v: 0.52

- *Console sostituita da piccole GUI*
- *Più facilità d'uso attraverso slider, bottoni e filtri*
- *Colore e direzione della luce cambiabile*



3.6.2012
v: 1.0

- *DHM25 e spot masaic per tutta la svizzera*
- *GUI per Textura*
- *GUI perOggetti*
- *Salvare impostazioni*
- *Help + Konsolenbefehle geupdatet*

Video 7

5 Codice sorgente

5.1 main.cpp

```
// includes
#include"win_UDF.h"
#include"d3d9_UDF.h"


// function prototypes
LRESULT CALLBACK WindowProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam); //processes window messages
void SubmitConsole(void);
void KeyProc(void);
void LoadINI(void);

// the entry point for any Windows program
int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine,
int nCmdShow)
{

        HWND TaskBar1, TaskBar2;                                //hide taskbar
        TaskBar1=FindWindow("Button","Start");                  //
        TaskBar2=FindWindow("Shell_TrayWnd","");                //
        ShowWindow(TaskBar1,SW_HIDE);                           //
        ShowWindow(TaskBar2,SW_HIDE);                           //


        DEVMODE dm;                                                     //get display settings
        dm.dmSize = sizeof(DEVMODE);                            //
        dm.dmDriverExtra = 0;                                   //
        EnumDisplaySettings(NULL, ENUM_CURRENT_SETTINGS, &dm);//

        SCREEN_WIDTH = dm.dmPelsWidth;  //save screen resolution
        SCREEN_HEIGHT = dm.dmPelsHeight;//

        GetCurrentDirectory(255,INI_DIR);                       //get ini dir
        sprintf(INI_DIR, "%s\\vars.ini", INI_DIR);      //

        WinCreate(hInstance, nCmdShow); //create window

        // set up and initialize Direct3D + Graphics
        initD3D(hWnd);
        LoadINI();      //load variable values out of ini


        //call some functions (not default -> CHANGE)
        HMap_DHM25(d3ddev, 1296);
        HMap_texture(d3ddev, "txt1");
        //FormsCreate(d3ddev);
        SkySphere_txt(d3ddev, "0");

        MSG msg; //Windows event messages
        while(TRUE)
        {
                while(PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)) //wait for next message
                {
                        // translate keystroke messages into the right format
                        TranslateMessage(&msg);
                        // send the message to WindowProc function
                        DispatchMessage(&msg);
                }

                if(msg.message == WM_QUIT || (m_KeyBuffer[DIK_ESCAPE] & 0x80)) //EXIT
                {
                        ShowCursor(TRUE); //precaution: if rotating camera while pressing esc
                        break;
                }

                fps();
                UpdateInput(); //get all pressed keys
```

```
                KeyProc(); //process keypess (^  / SPACE / ..)

                if (!Console.state) { //if console is not active
                        UpdateCamera();
                        GUIProc();
                        //update forms position
                        if (Forms.state && HMap.state) {FormUpdate(d3ddev);}
                } else {
                        ConsoleProc(); //translate into text for console
                        //check if console command is "exit" -> exit
                        if (UpToLow(Console.comand) == "exit") {break;}
                        //translate console command into action
                        if (Console.comand != "") {SubmitConsole();}
                }

                render_frame(); //render next frame
    }



        m_pDIKeyboardDevice->Unacquire();               //relase Keyboard
        m_pDIKeyboardDevice->Release();         //

        m_pDIMouseDevice->Unacquire();          //relase Mouse
        m_pDIMouseDevice->Release();            //

        m_pDIObject->Release();                 //relase Imput Object

        cleanD3D();// clean up DirectX and COM

        ShowWindow(TaskBar1,SW_SHOW);   //unhide taskbar
        ShowWindow(TaskBar2,SW_SHOW);   //

        return msg.wParam; // return this part of the WM_QUIT message to Windows
}


// main message handler
LRESULT CALLBACK WindowProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        char buffer[255];

        switch(message)
        {
        case WM_DESTROY: //exit
                {
                        PostQuitMessage(0);
                        return 0;
                } break;
                case WM_SETFOCUS: //get focus
                {
                        //reload input (cos: lose focus = lose input)
                        LoadInput (hWnd);
                } break;
                case WM_CHILDMSG: //message from GUI
                {
                        switch(wParam) {
                                case HM_DHM25:  //load DHM25
                                {
                                        HMap_DHM25(d3ddev, (int) lParam);
                                } break;
                                case HM_DHM200: //load DHM200
                                {
                                        HMap_DHM200(d3ddev, (int) lParam);
                                } break;
                                case HM_File:   //load File
                                {

                                } break;
                                case HM_DIV:    //set H-Map divider
                                {
                                        HMAP_DIV = (int) lParam;
                                        HMap_reload(d3ddev);
                                        sprintf(buffer, "%d", lParam);
                                        WritePrivateProfileString("HMAP", "HMAP_DIV", buffer, INI_DIR);
                                } break;
```

```
case HM_Texture:// set H-Map Texture
{
        switch(lParam) {
                case TX_GRASS:
                {
                        HMap_texture(d3ddev, "grass");
                } break;
                case TX_SNOW:
                {
                        HMap_texture(d3ddev, "snow");
                } break;
                case TX_TXT1:
                {
                        HMap_texture(d3ddev, "txt1");
                } break;
                case TX_TXT2:
                {
                        HMap_texture(d3ddev, "txt2");
                } break;
                case TX_TXT3:
                {
                        HMap_texture(d3ddev, "txt3");
                } break;
                case TX_SPOT:
                {
                        HMap_texture(d3ddev, "spot");
                } break;
        }
} break;
case HM_Texture_flt:    //toggle texture filter & antialiasing
{
        Toggle_Txt_Filter();
} break;
case EV_Skysphere:      //set skysphere
{
        char buffer[255];
        sprintf(buffer, "%d", lParam);
        SkySphere_txt(d3ddev, buffer);
} break;
case EV_CLRCLR:         //change clear-color
{
        CLRCOLOR = 0xFF000000+lParam;
        sprintf(buffer, "0x%X", 0xFF000000+lParam);
        WritePrivateProfileString("COLOR", "CLRCOLOR", buffer, INI_DIR);
} break;
case EV_LIGHTCLR:       //change light-color
{
        LoadLight_Diffuse(0xFF000000+lParam);
        sprintf(buffer, "0x%X", 0xFF000000+lParam);
        WritePrivateProfileString("LIGHT", "LIGHTCOLOR", buffer, INI_DIR);
} break;
case EV_LIGHTDIR:       //change light direction (0-360)
{
        LoadLight_Direction(lParam);
        sprintf(buffer, "%d", lParam);
        WritePrivateProfileString("LIGHT", "LIGHTDIR", buffer, INI_DIR);
} break;
case EV_LIGHTSTEEP: //change light steepness (0-90)
{
        LoadLight_steep((float)lParam*PI/180);
        sprintf(buffer, "%d", lParam);
        WritePrivateProfileString("LIGHT", "LIGHTSTEEPNESS", buffer, INI_DIR);
} break;
case OB_RESTITUTION:    //object restitution in %
{
        Forms.k = (float)lParam/100;
        sprintf(buffer, "%0.2f", (float)lParam/100);
        WritePrivateProfileString("OBJECTS", "RESTITUTION", buffer, INI_DIR);
} break;
case OB_FRICTION:               //object friction
{
        Forms.l = (float)lParam / 100;
        sprintf(buffer, "%0.2f", (float)lParam/100);
        WritePrivateProfileString("OBJECTS", "FRICTION", buffer, INI_DIR);
} break;
case OB_RESET:  //reset all objects
```

```
                                {
                                        FormsReset(d3ddev);
                                } break;
                                case OB_TIME:
                                {
                                        Forms.t_mult = (float)lParam/10;
                                        sprintf(buffer, "%0.1f", (float)lParam/10);
                                        WritePrivateProfileString("OBJECTS", "TIME", buffer, INI_DIR);
                                } break;
                        }
                }
        }

        return DefWindowProc (hWnd, message, wParam, lParam);//handle remaining messages
}

void KeyProc(void) {
        staticbool console_tog = 0; //changing between console on/off
        staticbool time_tog = 0; //changing between time on/off

        POINT cursorPos;
        GetCursorPos(&cursorPos);

        if ((m_KeyBuffer[0x0D] & 0x80) && console_tog == 0) { //if "^" was pressed
                console_tog = 1;
                Console.state = !(Console.state);
        }
        if (!(m_KeyBuffer[0x0D] & 0x80) && console_tog == 1) {
                console_tog = 0;
        }


        if ((m_KeyBuffer[0x39] & 0x80) && time_tog == 0) {      //if "SPACE" was pressed
                time_tog = 1;
                Forms.state = !(Forms.state);
        }
        if (!(m_KeyBuffer[0x39] & 0x80) && time_tog == 1) {
                time_tog = 0;
        }
}


void SubmitConsole (void) {

        std::string output = "";
        char buffer[255];
        long hex;

        //list to hold console parameter
        std::list<std::string> l_param;
        //split console comand into parameter
        split(Console.comand.substr( Console.comand.find('(') + 1, Console.comand.find(')') - Console.comand.find('(') -
        1),',', l_param);
        //string array to hold console parameter
        std::string* a_param = new std::string[l_param.size()];
        //counter
        int i = 0;
        //for all parameters
        for (std::list<std::string>::iterator it = l_param.begin(); it != l_param.end(); it++) {
                a_param[i] = rem_space((std::string) *it); //translate list into array (easier handling)
                i++;
        }


        if (UpToLow(Console.comand.substr(0, 5)) == "call ") { //suffix call (call a function)

                if (UpToLow(Console.comand.substr(5, 6)) == "dhm200") { //DHM200
                        HMAP_DIV = 200;
                        switch (HMap_DHM200(d3ddev, (int) atof(a_param[0].c_str()))) {
                                casetrue:
                                output = "CALL: Height-Map DHM200 -> Table Nr. "+a_param[0]+" was loaded successfully!";
                                break;
                                casefalse:
                                output = "CALL: Height-Map DHM200 -> Table Nr. "+a_param[0]+" could not be loaded!";
                                break;
                        }
```

```
        }

        if (UpToLow(Console.comand.substr(5, 5)) == "dhm25") {   //DHM25
                HMAP_DIV = 25;
                switch(HMap_DHM25(d3ddev, (int) atof(a_param[0].c_str()))) {
                        casetrue:
                        output = "CALL: Height-Map DHM25 -> Table Nr. "+a_param[0]+" was loaded successfully!";
                        break;
                        casefalse:
                        output = "CALL: Height-Map DHM25 -> Table Nr. "+a_param[0]+" could not be loaded!";
                        break;
                }
        }

        if (UpToLow(Console.comand.substr(5, 7)) == "hmaptxt") {//H-Map Texture

                if(a_param[0] == "spot") {
                        if(HMap_texture(d3ddev, "spot")) {
                        output = "CALL: Height-Map Texture -> File "+a_param[0]+" was loaded successfully!";
                        } else {
                        output = "CALL: Height-Map Texture -> File "+a_param[0]+" could not be loaded!";
                        }
                } else {
                        if(HMap_texture(d3ddev, a_param[0].c_str())) {
                        output = "CALL: Height-Map Texture -> File "+a_param[0]+" was loaded successfully!";
                        } else {
                        output = "CALL: Height-Map Texture -> File "+a_param[0]+" could not be loaded!";
                        }
                }
        }

        if (UpToLow(Console.comand.substr(5, 9)) == "skysphere") {      //SkySphere

                switch(SkySphere_txt(d3ddev, a_param[0].c_str())){
                        casetrue:
                                output = "CALL: SkySphere -> Skysphere "+a_param[0]+" was loaded successfully!";
                                break;
                        casefalse:
                                output = "CALL: SkySphere -> Skysphere "+a_param[0]+" could not be loaded!";
                                break;
                }
        }

} elseif (UpToLow(Console.comand.substr(0, 4)) == "get ") { //suffix get (get a variable)

        //camera defines
        if (UpToLow(Console.comand.substr(4, 8)) == "cam_sens") { //sensitivity
                sprintf(buffer,"%g",(float) SENSITIVITY);
                output = buffer;
                output = "GET: Camera sensitivity -> " + output;
        } elseif (UpToLow(Console.comand.substr(4, 9)) == "cam_speed") { //speed
                sprintf(buffer,"%g",(float) SPEED);
                output = buffer;
                output = "GET: Camera speed -> " + output;
        } elseif (UpToLow(Console.comand.substr(4, 9)) == "cam_wheel") { //wheel multip
                sprintf(buffer,"%g",(float) WHEEL_MULTIP);
                output = buffer;
                output = "GET: Camera Wheel multiplicator -> " + output;
        } elseif (UpToLow(Console.comand.substr(4, 13)) == "cam_shift_mov") { //shift movement multip
                sprintf(buffer,"%g",(float) SHIFT_MOV_MULTIP);
                output = buffer;
                output = "GET: Camera Shift movement multiplicator -> " + output;
        } elseif (UpToLow(Console.comand.substr(4, 13)) == "cam_shift_rot") { //shift rotation multip
                sprintf(buffer,"%g",(float) SHIFT_ROT_MULTIP);
                output = buffer;
                output = "GET: Camera Shift rotation multiplicator -> " + output;
        } elseif (UpToLow(Console.comand.substr(4, 12)) == "cam_ctrl_mov") { //ctrl movement multip
                sprintf(buffer,"%g",(float) CTRL_MOV_MULTIP);
                output = buffer;
                output = "GET: Camera CTRL movement multiplicator -> " + output;
        } elseif (UpToLow(Console.comand.substr(4, 12)) == "cam_ctrl_rot") { //ctrl rotation multip
                sprintf(buffer,"%g",(float) CTRL_ROT_MULTIP);
                output = buffer;
                output = "GET: Camera CTRL rotation multiplicator -> " + output;
        }
```

```
//color defines
if (UpToLow(Console.comand.substr(4, 8)) == "clrcolor") { //clear color
        sprintf(buffer,"0x%X", CLRCOLOR);
        output = buffer;
        output = "GET: Clear color -> " + output;
} elseif (UpToLow(Console.comand.substr(4, 16)) == "consolefontcolor") { //fontcolor(input) color
        sprintf(buffer,"0x%X", FONTCOLOR_CONSOLE);
        output = buffer;
        output = "GET: Console font color -> " + output;
}

//light defines
if (UpToLow(Console.comand.substr(4, 8)) == "lightdir") { //light direction
        sprintf(buffer,"%.0f°", atan2(light.Direction.z, light.Direction.x)*180/PI+90 );
        output = buffer;
        output = "GET: Light Direction -> " + output;
} elseif (UpToLow(Console.comand.substr(4, 14)) == "lightsteepness") { //light steepness
        sprintf(buffer,"%.0f°", -atan(light.Direction.y/(pow( (float) pow( (float) light.Direction.x,
        (float) 2 ) + pow( (float) light.Direction.z, (float) 2 ) , (float) 0.5 )))*180/PI);
        output = buffer;
        output = "GET: Light Steepness -> " + output;
} elseif (UpToLow(Console.comand.substr(4, 10)) == "lightcolor") { //light color
        sprintf(buffer,"0x%02X%02X%02X", (int)(light.Diffuse.r*255+0.5), (int)(light.Diffuse.g*255+0.5),
        (int)(light.Diffuse.b*255+0.5));
        output = buffer;
        output = "GET: Light Color -> " + output;
}

//global defines
if (UpToLow(Console.comand.substr(4, 12)) == "screen_width") { //screen width
        sprintf(buffer,"%d", SCREEN_WIDTH);
        output = buffer;
        output = "GET: Screen width -> " + output;
} elseif (UpToLow(Console.comand.substr(4, 13)) == "screen_height") { //screen height
        sprintf(buffer,"%d", SCREEN_HEIGHT);
        output = buffer;
        output = "GET: Screen height -> " + output;
}

//hmap defines
if (UpToLow(Console.comand.substr(4, 12)) == "hmap_width") { //hmap width
        sprintf(buffer,"%d", HMAP_WIDTH);
        output = buffer;
        output = "GET: Height-Map width -> " + output;
} elseif (UpToLow(Console.comand.substr(4, 11)) == "hmap_height") { //hmap size
        sprintf(buffer,"%d", HMAP_HEIGHT);
        output = buffer;
        output = "GET: Height-Map height -> " + output;
} elseif (UpToLow(Console.comand.substr(4, 11)) == "hmap_size") { //hmap size
        sprintf(buffer,"%d", HMAP_SIZE);
        output = buffer;
        output = "GET: Height-Map size -> " + output;
} elseif (UpToLow(Console.comand.substr(4, 8)) == "hmap_div") { //hmap div
        sprintf(buffer,"%g", HMAP_DIV);
        output = buffer;
        output = "GET: Height-Map divider -> " + output;
}

//win defines

if (UpToLow(Console.comand.substr(4, 7)) == "win_tit") { //window title
        output = WIN_TIT;
        output = "GET: Window title -> \"" + output + "\"";
}

//objects defines
if (UpToLow(Console.comand.substr(4, 8)) == "form_res") { //window title
        sprintf(buffer,"%.2f", Forms.k);
        output = buffer;
        output = "GET: Coefficient of restitution -> " + output;
} elseif (UpToLow(Console.comand.substr(4, 9)) == "form_fric") { //window title
        sprintf(buffer,"%.2f", Forms.l);
        output = buffer;
        output = "GET: Coefficient of friction -> " + output;
}
```

```
} elseif (UpToLow(Console.comand.substr(0, 4)) == "set ") { //suffix set (set a variable)

	if (UpToLow(Console.comand.substr(4, 8)) == "cam_sens") { //sensitivity
		output = Console.comand.substr(13);
		SENSITIVITY = (float) atof(output.c_str());
		WritePrivateProfileString("CAMERA", "SENSITIVITY", output.c_str(), INI_DIR);
		output = "SET: Camera sensitivity -> " + output;
	} elseif (UpToLow(Console.comand.substr(4, 9)) == "cam_speed") { //speed
		output = Console.comand.substr(14);
		SPEED = (float) atof(output.c_str());
		WritePrivateProfileString("CAMERA", "SPEED", output.c_str(), INI_DIR);
		output = "SET: Camera speed -> " + output;
	} elseif (UpToLow(Console.comand.substr(4, 9)) == "cam_wheel") { //wheel multip
		output = Console.comand.substr(14);
		WHEEL_MULTIP = (float) atof(output.c_str());
		WritePrivateProfileString("CAMERA", "WHEEL_MULTIP", output.c_str(), INI_DIR);
		output = "SET: Camera wheel multiplicator -> " + output;
	} elseif (UpToLow(Console.comand.substr(4, 13)) == "cam_shift_mov") { //shift movement multip
		output = Console.comand.substr(18);
		SHIFT_MOV_MULTIP = (float) atof(output.c_str());
		WritePrivateProfileString("CAMERA", "SHIFT_MOV_MULTIP", output.c_str(), INI_DIR);
		output = "SET: Camera Shift movement multiplicator -> " + output;
	} elseif (UpToLow(Console.comand.substr(4, 13)) == "cam_shift_rot") { //shift rotation multip
		output = Console.comand.substr(18);
		SHIFT_ROT_MULTIP = (float) atof(output.c_str());
		WritePrivateProfileString("CAMERA", "SHIFT_ROT_MULTIP", output.c_str(), INI_DIR);
		output = "SET: Camera Shift rotation multiplicator -> " + output;
	} elseif (UpToLow(Console.comand.substr(4, 12)) == "cam_ctrl_mov") { //ctrl movement multip
		output = Console.comand.substr(17);
		CTRL_MOV_MULTIP = (float) atof(output.c_str());
		WritePrivateProfileString("CAMERA", "CTRL_MOV_MULTIP", output.c_str(), INI_DIR);
		output = "SET: Camera CTRL movement multiplicator -> " + output;
	} elseif (UpToLow(Console.comand.substr(4, 12)) == "cam_ctrl_rot") { //ctrl rotation multip
		output = Console.comand.substr(17);
		CTRL_ROT_MULTIP = (float) atof(output.c_str());
		WritePrivateProfileString("CAMERA", "CTRL_ROT_MULTIP", output.c_str(), INI_DIR);
		output = "SET: Camera CTRL rotation multiplicator -> " + output;
	}

	//color defines
	if (UpToLow(Console.comand.substr(4, 8)) == "clrcolor") { //clear color
		output = Console.comand.substr(15);
		sscanf_s(output.c_str(), "%x", &hex);
		CLRCOLOR = (D3DCOLOR) (float) hex;
		WritePrivateProfileString("COLOR", "CLRCOLOR", ("0x"+output).c_str(), INI_DIR);
		output = "SET: Clear color -> 0x" + LowToUp(output);
	} elseif (UpToLow(Console.comand.substr(4, 16)) == "consolefontcolor") { //fontcolor(input) color
		output = Console.comand.substr(23);
		sscanf_s(output.c_str(), "%x", &hex);
		FONTCOLOR_CONSOLE = (D3DCOLOR) (float) hex;
		WritePrivateProfileString("COLOR", "FONTCOLOR_CONSOLE", ("0x"+output).c_str(), INI_DIR);
		output = "SET: Console font color -> 0x" + LowToUp(output);
	}

	//light defines
	if (UpToLow(Console.comand.substr(4, 8)) == "lightdir") { //light direction
		output = Console.comand.substr(13);
		LoadLight_Direction((int)atoi(output.c_str()));
		WritePrivateProfileString("LIGHT", "LIGHTDIR", output.c_str(), INI_DIR);
		output = "SET: Light Direction -> " + output;
	} elseif (UpToLow(Console.comand.substr(4, 14)) == "lightsteepness") { //light steepness
		output = Console.comand.substr(19);
		LoadLight_steep(atof(output.c_str())*PI/180);
		WritePrivateProfileString("LIGHT", "LIGHTSTEEPNESS", output.c_str(), INI_DIR);
		output = "SET: Light Steepness -> " + output;
	} elseif (UpToLow(Console.comand.substr(4, 10)) == "lightcolor") { //light color
		output = Console.comand.substr(17);
		sscanf_s(output.c_str(), "%x", &hex);
		LoadLight_Diffuse(0xFF000000+hex);
		WritePrivateProfileString("LIGHT", "LIGHTCOLOR", ("0x"+output).c_str(), INI_DIR);
		output = "SET: Light Color -> 0x" + LowToUp(output);
	}

	//hmapdefines
```

```
		if (UpToLow(Console.comand.substr(4, 8)) == "hmap_div") { //hmap div
			output = Console.comand.substr(13);
			HMAP_DIV = (float) atof(output.c_str());
			HMap_reload(d3ddev);
			WritePrivateProfileString("HMAP", "HMAP_DIV", output.c_str(), INI_DIR);
			output = "SET: Height-Map divider -> " + output;
		}


		//objects defines
		if (UpToLow(Console.comand.substr(4, 8)) == "form_res") { //window title
			output = Console.comand.substr(13);
			Forms.k = (float) atof(output.c_str());
			WritePrivateProfileString("OBJECTS", "RESTITUTION", output.c_str(), INI_DIR);
			output = "SET: Coefficient of restitution -> " + output;
		} elseif (UpToLow(Console.comand.substr(4, 9)) == "form_fric") { //window title
			output = Console.comand.substr(14);
			Forms.l = (float) atof(output.c_str());
			WritePrivateProfileString("OBJECTS", "FRICTION", output.c_str(), INI_DIR);
			output = "SET: Coefficient of friction -> " + output;
		}


	} elseif(UpToLow(Console.comand.substr(0, 4)) == "help") {//no suffix
		STARTUPINFO          si = { sizeof(si) };
		PROCESS_INFORMATION  pi;
		CreateProcess("Help\\Help.exe", NULL, NULL, NULL, FALSE, NULL, NULL, NULL, &si, &pi);
		output = "Help was started.";
	}

	for (int i = 0; i <=1; i++)     {
		Console.output[i] = Console.output[i+1];//shift outputs 1 up
	}

	if (output != "") {
		Console.output[2] = output;
	} else {
		Console.output[2] = "ERROR: Command \"" + Console.comand + "\" could not be executed!";
	}

	Console.comand = "";
}

void LoadINI(void) { //read and set all variables writen in the ini file
	char buffer[255];
	long hex;
	DWORD size = 500;

	GetPrivateProfileString("CAMERA","SENSITIVITY","0.001", buffer, size, INI_DIR);
	SENSITIVITY = (float) atof(buffer);
	GetPrivateProfileString("CAMERA","SPEED","0.3", buffer, size, INI_DIR);
	SPEED = (float) atof(buffer);
	GetPrivateProfileString("CAMERA","WHEEL_MULTIP","7", buffer, size, INI_DIR);
	WHEEL_MULTIP = (float) atof(buffer);
	GetPrivateProfileString("CAMERA","SHIFT_MOV_MULTIP","3", buffer, size, INI_DIR);
	SHIFT_MOV_MULTIP = (float) atof(buffer);
	GetPrivateProfileString("CAMERA","SHIFT_ROT_MULTIP","1.5", buffer, size, INI_DIR);
	SHIFT_ROT_MULTIP = (float) atof(buffer);
	GetPrivateProfileString("CAMERA","CTRL_MOV_MULTIP","0.4", buffer, size, INI_DIR);
	CTRL_MOV_MULTIP = (float) atof(buffer);
	GetPrivateProfileString("CAMERA","CTRL_ROT_MULTIP","0.8", buffer, size, INI_DIR);
	CTRL_ROT_MULTIP = (float) atof(buffer);

	GetPrivateProfileString("COLOR","CLRCOLOR","0xFF2A2A2A", buffer, size, INI_DIR);
	sscanf_s(buffer, "%x", &hex);
	CLRCOLOR = (D3DCOLOR) (float) hex;
	GetPrivateProfileString("COLOR","FONTCOLOR_CONSOLE","0xFF999999", buffer, size, INI_DIR);
	sscanf_s(buffer, "%x", &hex);
	FONTCOLOR_CONSOLE = (D3DCOLOR) (float) hex;

	GetPrivateProfileString("LIGHT","LIGHTDIR","0", buffer, size, INI_DIR);
	LoadLight_Direction((int) atoi(buffer));
	GetPrivateProfileString("LIGHT","LIGHTSTEEPNESS","16", buffer, size, INI_DIR);
	LoadLight_steep((int) atoi(buffer));
	GetPrivateProfileString("LIGHT","LIGHTCOLOR","0xFFB2B2B2", buffer, size, INI_DIR);
	sscanf_s(buffer, "%x", &hex);
```

```
        LoadLight_Diffuse((D3DCOLOR) (float) hex);

        GetPrivateProfileString("HMAP","HMAP_DIV","25", buffer, size, INI_DIR);
        HMAP_DIV = (float) atof(buffer);

        GetPrivateProfileString("OBJECTS","RESTITUTION","0.01", buffer, size, INI_DIR);
        Forms.k = (float) atof(buffer);
        GetPrivateProfileString("OBJECTS","FRICTION","0.9", buffer, size, INI_DIR);
        Forms.l = (float) atof(buffer);
        GetPrivateProfileString("OBJECTS","TIME","1", buffer, size, INI_DIR);
        Forms.t_mult = (float) atof(buffer);

}
```

5.2 win_UDF.h

```
#ifndef win_UDF
#define win_UDF
#include<windows.h>
#include<windowsx.h>
#include"win_defines.h"

//function prototypes
void WinCreate(HINSTANCE hInstance, int nCmdShow); //create the window
LRESULT CALLBACK WindowProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam); //processes window messages

// global declarations
HWND hWnd;// the handle for the window, filled by a function
WNDCLASSEX wc;// this struct holds information for the window class

void WinCreate(HINSTANCE hInstance, int nCmdShow) {

        ZeroMemory(&wc, sizeof(WNDCLASSEX)); // clear out the window class for use

        // fill in the struct with the needed information
        wc.cbSize = sizeof(WNDCLASSEX);
        wc.style = CS_HREDRAW | CS_VREDRAW;
        wc.lpfnWndProc = WindowProc;
        wc.hInstance = hInstance;
        wc.hCursor = LoadCursor(NULL, IDC_ARROW);
        wc.hbrBackground = (HBRUSH)COLOR_WINDOW;
        wc.lpszClassName = "WindowClass1";

        RegisterClassEx(&wc);// register the window class

        // create the window and use the result as the handle
        hWnd = CreateWindowEx(NULL,
"WindowClass1",    // name of the window class
                        WIN_TIT,   // title of the window
                        WIN_STYLE,    // window style
                        0, 0,   // position of the window
                        SCREEN_WIDTH, SCREEN_HEIGHT,   // size of the window
                        NULL,    // we have no parent window, NULL
                        NULL,    // we aren't using menus, NULL
                        hInstance,    // application handle
                        NULL);    // used with multiple windows, NULL


        ShowWindow(hWnd, nCmdShow);// display the window on the screen

}


#endif
```

5.3 d3d9_UDF.h

```
#ifndef d3d9_UDF
#define d3d9_UDF

#define DIRECTINPUT_VERSION     0x0800

// includes
#include<d3d9.h>
#include<d3dx9.h>
#include<math.h>
#include"camera.h"
#include"console.h"
#include"info.h"
#include"form.h"
#include"env.h"
#include"gui.h"
#include"global_defines.h"

// global declarations
LPDIRECT3D9 d3d;    // the pointer to our Direct3D interface
LPDIRECT3DDEVICE9 d3ddev;    // the pointer to the device class
LPD3DXFONT c_font;      //font for console text
D3DLIGHT9 light;    // create the light struct
D3DMATERIAL9 material;    // create the material struct

//function prototypes (D3D9)
void initD3D(HWND hWnd);    // sets up and initializes Direct3D
void initGraphics(void);//sets up and initializes Graphics
void LoadLight();//sets up and initializes Light
void LoadLight_Direction(int deg);                      //changes the lights direction
void LoadLight_steep(float steep);                      //changes the lights steepness
void LoadLight_Diffuse(D3DXCOLOR color);                //changes the lights color

void Toggle_Txt_Filter(void);


void render_frame(void);        // renders a single frame
void cleanD3D(void);            // closes Direct3D and releases memory


// this function initializes and prepares Direct3D for use
void initD3D(HWND hWnd)
{
        d3d = Direct3DCreate9(D3D_SDK_VERSION);    // create the Direct3D interface

        D3DPRESENT_PARAMETERS d3dpp;    // create a struct to hold various device information

        ZeroMemory(&d3dpp, sizeof(d3dpp));    // clear out the struct for use
        d3dpp.Windowed = TRUE;    // program windowed, not fullscreen
        d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;    // discard old frames
        //d3dpp.MultiSampleType = D3DMULTISAMPLE_2_SAMPLES;
        d3dpp.hDeviceWindow = hWnd;    // set the window to be used by Direct3D
        d3dpp.BackBufferFormat = D3DFMT_X8R8G8B8;    // set the back buffer format to 32-bit
        d3dpp.BackBufferWidth = SCREEN_WIDTH;    // set the width of the buffer
        d3dpp.BackBufferHeight = SCREEN_HEIGHT;    // set the height of the buffer
        d3dpp.EnableAutoDepthStencil = TRUE;    // automatically run the z-buffer for us
        d3dpp.AutoDepthStencilFormat = D3DFMT_D16;    // 16-bit pixel format for the z-buffer

        // create a device class using this information and the info from the d3dpp stuct
        d3d->CreateDevice(D3DADAPTER_DEFAULT,
                          D3DDEVTYPE_HAL,
                          hWnd,
                          D3DCREATE_SOFTWARE_VERTEXPROCESSING,
&d3dpp,
&d3ddev);


        D3DXCreateFont(d3ddev, 25, 0, 0, 0, FALSE, DEFAULT_CHARSET, OUT_DEFAULT_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH |
        FF_DONTCARE, TEXT("Tahoma"), &c_font );

        d3ddev->SetRenderState(D3DRS_LIGHTING, TRUE);    // turn on the 3D lighting
        d3ddev->SetRenderState(D3DRS_CULLMODE, D3DCULL_NONE);    // both sides of the triangles
        d3ddev->SetRenderState(D3DRS_ZENABLE, TRUE);    // turn on the z-buffer
```

```
        d3ddev->SetRenderState(D3DRS_ALPHABLENDENABLE, TRUE);    // turn on the color blending
        d3ddev->SetRenderState(D3DRS_SRCBLEND, D3DBLEND_SRCALPHA);    // set source factor
        d3ddev->SetRenderState(D3DRS_DESTBLEND, D3DBLEND_INVSRCALPHA);    // set dest factor
        d3ddev->SetRenderState(D3DRS_BLENDOP, D3DBLENDOP_ADD);    // set the operation
        d3ddev->SetRenderState(D3DRS_AMBIENT, D3DCOLOR_XRGB(100, 100, 100));    // ambient light
        d3ddev->SetRenderState(D3DRS_MULTISAMPLEANTIALIAS, TRUE);    //full scene antialiasing

        d3ddev->SetSamplerState(0, D3DSAMP_MAXANISOTROPY, 8);    // anisotropic level
        d3ddev->SetSamplerState(0, D3DSAMP_MINFILTER, D3DTEXF_ANISOTROPIC);    // minification
        d3ddev->SetSamplerState(0, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR);    // magnification
        d3ddev->SetSamplerState(0, D3DSAMP_MIPFILTER, D3DTEXF_LINEAR);    // mipmap

        LoadInput (hWnd);       //initialize keyboard & mouse input
        LoadLight(); // call the function to initialize the light and material
        initGraphics(); //init graphics
}



voidinitGraphics() {
        GUIInit(d3ddev);                //init graphic user interface
        ConsoleInit(d3ddev);            //init console
        InfoInit();                     //init info text (upper right corner)
        initHMap(d3ddev);               //init heightmap
        FormInit();                     //init forms
        SkySphere_init(d3ddev);         //init skybox whit empty textuer
        SkySphere_txt(d3ddev, "0");     //
}



// this is the function used to render a single frame
void render_frame(void)
{

        d3ddev->Clear(0, NULL, D3DCLEAR_TARGET, CLRCOLOR, 1.0f, 0);
        d3ddev->Clear(0, NULL, D3DCLEAR_ZBUFFER, CLRCOLOR, 1.0f, 0);

        d3ddev->BeginScene();

        //*************** 3D ***************

        d3ddev->SetFVF(FVF_3D); // select which vertex format we are using

        D3DXMATRIX matTranslate, matRotate, matScale; //matrices for world transform
        D3DXMatrixTranslation(&matTranslate, 0.0f, 0.0f, 0.0f); //create "reset" matrix
        d3ddev->SetTransform(D3DTS_WORLD, &matTranslate); //reset world transform

        D3DXMATRIX matView;    // the view transform matrix
        D3DXMatrixLookAtLH(&matView, &m_vEyePt, &m_vLookAtPt, &m_vUp);    // camera position
        d3ddev->SetTransform(D3DTS_VIEW, &matView);    // set the view transform to matView

        D3DXMATRIX matProjection;     // the projection transform matrix
        D3DXMatrixPerspectiveFovLH(&matProjection, D3DXToRadian(45),
        (FLOAT)SCREEN_WIDTH / (FLOAT)SCREEN_HEIGHT, 1.0f, 5000.0f);
        d3ddev->SetTransform(D3DTS_PROJECTION, &matProjection);    // set the projection

        //--- Draw SkySphere ---
        d3ddev->SetRenderState(D3DRS_AMBIENT, D3DCOLOR_XRGB(255, 255, 255));     //max light -> no shadows
        D3DXMatrixScaling(&matScale, 10000.0f, 10000.0f, 10000.0f);              //resize (1000 px)
        D3DXMatrixTranslation(&matTranslate, 0, -200, 0);       // move 200 down
        d3ddev->SetTransform(D3DTS_WORLD, &(matScale*matTranslate)); //submit matrices
        d3ddev->SetMaterial(&HMap.material);    //set material
        d3ddev->SetTexture(0, Env.skysphere_txt);       //set texuter
        Env.skysphere->DrawSubset(0);                           //draw skysphere
        d3ddev->SetRenderState(D3DRS_AMBIENT, D3DCOLOR_XRGB(100, 100, 100));    //reset light to normal

        //--- Draw HMap ---
        D3DXMatrixTranslation(&matTranslate, 0.0f, 0.0f, 0.0f); //create "reset" matrix
        d3ddev->SetTransform(D3DTS_WORLD, &matTranslate); //reset world transform
        if (HMap.state == 1) {
                d3ddev->SetMaterial(&HMap.material);    //select the material
                d3ddev->SetTexture(0, HMap.texture);    //select the textuer
                d3ddev->SetStreamSource(0, HMap.v_buffer, 0, sizeof(VERTEX_3D));    // select the vertex buffer
                d3ddev->SetIndices(HMap.i_buffer);      // select the index buffer
                d3ddev->DrawIndexedPrimitive(D3DPT_TRIANGLESTRIP, 0, 0, HMap.size, 0, HMap.size*2-HMAP_WIDTH*4);//draw hmap
```

```
}

//--- Draw Form ---
for(int i=0; i<Forms.count; i++) {
	D3DXMatrixRotationAxis(&matRotate, &Forms.Forms[i].RotAx, Forms.Forms[i].RotDeg); //rotate mesh
	D3DXMatrixTranslation(&matTranslate, Forms.Forms[i].x, Forms.Forms[i].y, Forms.Forms[i].z); //move mesh
	d3ddev->SetTransform(D3DTS_WORLD, &(matRotate*matTranslate));	//submit rotation + movement
	for(DWORD j = 0; j < Forms.Forms[i].numMaterials; j++)	// loop through each subset
	{
		d3ddev->SetMaterial(&Forms.Forms[i].material[j]);	// set the material for the subset
		d3ddev->SetTexture(0, Forms.Forms[i].texture[j]);	// set the texture
		Forms.Forms[i].mesh->DrawSubset(j);	// draw the subset
	}
}

//*************** 2D ***************
d3ddev->SetTexture(0, NULL);
d3ddev->SetFVF(FVF_2D);// select which vertex format we are using

//--- GUI ---

/*d3ddev->SetTexture(0, GUI.tBkg);
d3ddev->SetStreamSource(0, GUI.bBkg, 0, sizeof(VERTEX_2D)); // select the vertex buffer to display
d3ddev->DrawPrimitive(D3DPT_TRIANGLESTRIP, 0, 2); // copy the vertex buffer to the back buffer*/

d3ddev->SetTexture(0, GUI.tTb1);
d3ddev->SetStreamSource(0, GUI.bTb1, 0, sizeof(VERTEX_2D)); // select the vertex buffer to display
d3ddev->DrawPrimitive(D3DPT_TRIANGLESTRIP, 0, 2); // copy the vertex buffer to the back buffer

d3ddev->SetTexture(0, GUI.tTb2);
d3ddev->SetStreamSource(0, GUI.bTb2, 0, sizeof(VERTEX_2D)); // select the vertex buffer to display
d3ddev->DrawPrimitive(D3DPT_TRIANGLESTRIP, 0, 2); // copy the vertex buffer to the back buffer

d3ddev->SetTexture(0, GUI.tTb3);
d3ddev->SetStreamSource(0, GUI.bTb3, 0, sizeof(VERTEX_2D)); // select the vertex buffer to display
d3ddev->DrawPrimitive(D3DPT_TRIANGLESTRIP, 0, 2); // copy the vertex buffer to the back buffer

d3ddev->SetTexture(0, GUI.tTb4);
d3ddev->SetStreamSource(0, GUI.bTb4, 0, sizeof(VERTEX_2D)); // select the vertex buffer to display
d3ddev->DrawPrimitive(D3DPT_TRIANGLESTRIP, 0, 2); // copy the vertex buffer to the back buffer


//--- Console ---
if (Console.state) {
	//input
	c_font->DrawText(NULL, (LPCSTR) (Console.text + "_").c_str(), -1, &Console.rect4, 0, FONTCOLOR_CONSOLE);
	//output 1 line
	c_font->DrawText(NULL, (LPCSTR) Console.output[0].c_str(), -1, &Console.rect1, 0, FONTCOLOR_CONSOLE-
	0xCC000000);
	//output 2 line
	c_font->DrawText(NULL, (LPCSTR) Console.output[1].c_str(), -1, &Console.rect2, 0, FONTCOLOR_CONSOLE-
	0x77000000);
	//output 3 line
	c_font->DrawText(NULL, (LPCSTR) Console.output[2].c_str(), -1, &Console.rect3, 0, FONTCOLOR_CONSOLE-
	0x00000000);

	d3ddev->SetTexture(0, Console.texture);
	d3ddev->SetStreamSource(0, Console.buffer, 0, sizeof(VERTEX_2D)); // select the vertex buffer to display
	d3ddev->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 2); // copy the vertex buffer to the back buffer
}


//--- Info ---
char buffer[255];
sprintf(buffer,"World Beta Version: 1.0");
c_font->DrawText(NULL, buffer, -1, &Info.line1, 0, FONTCOLOR_CONSOLE);
sprintf(buffer,"Developer: Marco Ruggia");
c_font->DrawText(NULL, buffer, -1, &Info.line2, 0, FONTCOLOR_CONSOLE);
sprintf(buffer,"X:%g Y:%g Z:%g", floor(m_vEyePt.x+0.5), floor(m_vEyePt.z+0.5), floor(m_vEyePt.y+0.5));
c_font->DrawText(NULL, buffer, -1, &Info.line3, 0, FONTCOLOR_CONSOLE);
sprintf(buffer,"FPS:%g", floor((FPS[0]*100)+0.5)/100);
c_font->DrawText(NULL, buffer, -1, &Info.line4, 0, FONTCOLOR_CONSOLE);

d3ddev->EndScene();

d3ddev->Present(NULL, NULL, NULL, NULL);
```

```
}


// this is the function that cleans up Direct3D and COM
void cleanD3D(void)
{
        d3ddev->Release();    // close and release the 3D device
        d3d->Release();    // close and release Direct3D
}


void fps(void) {
        staticint last_time = GetTickCount();

        FPS[4] = FPS[3];
        FPS[3] = FPS[2];
        FPS[2] = FPS[1];
        FPS[1] = 1.0f/((GetTickCount()-last_time)/1000.0f);
        FPS[0] = (FPS[1] + FPS[2] + FPS[3] + FPS[4])/4;

        last_time = GetTickCount();
}

// this is the function that sets up the lights and materials
void LoadLight()
{
        ZeroMemory(&light, sizeof(light));    // clear out the light struct for use
        light.Type = D3DLIGHT_DIRECTIONAL;    // make the light type 'directional light'
        light.Diffuse = D3DXCOLOR(0.7f, 0.7f, 0.7f, 1.0f);    // set the light's color
        light.Direction = D3DXVECTOR3(cos(PI/2), -0.3f, -sin(PI/2));

        d3ddev->SetLight(0, &light);    // send the light struct properties to light #0
        d3ddev->LightEnable(0, TRUE);    // turn on light #0

        ZeroMemory(&material, sizeof(D3DMATERIAL9));    // clear out the struct for use
        material.Diffuse = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f);    // set diffuse color to white
        material.Ambient = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f);    // set ambient color to white

        d3ddev->SetMaterial(&material);    // set the globably-used material to &material
}


// this is the function that changes the lights direction
void LoadLight_Direction(int deg)
{
        D3DXVECTOR3 direction = D3DXVECTOR3(cos( ((float)deg-90)*PI/180 ), 0.0f, sin( ((float)deg-90)*PI/180 ));

        light.Direction.x = direction.x;//set the light's direction
        light.Direction.z = direction.z;//
    d3ddev->SetLight(0, &light);    // send the light struct properties to light
}




// this is the function that changes the lights steepness
void LoadLight_steep(float steep)
{
        light.Direction.y = tan((float) -steep) * pow( (float) pow( (float) light.Direction.x, (float) 2 ) + pow( (float)
        light.Direction.z, (float) 2 ) , (float) 0.5 );
        d3ddev->SetLight(0, &light);
}

// this is the function that changes the lights color
void LoadLight_Diffuse(D3DXCOLOR color)
{
        light.Diffuse = color;    // set the light's color
        d3ddev->SetLight(0, &light);    // send the light struct properties to light
}

void Toggle_Txt_Filter(void) {
        staticbool toggle = 1;

        if (toggle == 1) {
                d3ddev->SetSamplerState(0, D3DSAMP_MAXANISOTROPY, 1);    // anisotropic level
```

```
                d3ddev->SetSamplerState(0, D3DSAMP_MINFILTER, NULL);    // minification
                d3ddev->SetSamplerState(0, D3DSAMP_MAGFILTER, NULL);    // magnification
                d3ddev->SetSamplerState(0, D3DSAMP_MIPFILTER, NULL);    // mipmap
                d3ddev->SetRenderState(D3DRS_MULTISAMPLEANTIALIAS, FALSE);    //full scene antialiasing
        } else {
                d3ddev->SetSamplerState(0, D3DSAMP_MAXANISOTROPY, 8);    // anisotropic level
                d3ddev->SetSamplerState(0, D3DSAMP_MINFILTER, D3DTEXF_ANISOTROPIC);    // minification
                d3ddev->SetSamplerState(0, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR);    // magnification
                d3ddev->SetSamplerState(0, D3DSAMP_MIPFILTER, D3DTEXF_LINEAR);    // mipmap
                d3ddev->SetRenderState(D3DRS_MULTISAMPLEANTIALIAS, TRUE);    //full scene antialiasing
        }

        toggle = !toggle;
}


#endif
```

5.4 camera.h

```
#ifndef CAMERA
#define CAMERA

#include<d3d9.h>
#include<d3dx9.h>
#include"camera_defines.h"
#include"input.h"

bool UpdateCamera (void);
void RotateCamera (void);
void MoveCamera (void);

D3DXVECTOR3             m_vEyePt (450.0f, 150.0f, 0.0f);     //camera positino
//D3DXVECTOR3           m_vEyePt (200.0f, 150.0f, 240.0f);     //camera positino
D3DXVECTOR3             m_vLookAtPt (400.0f, 0.0f, 400.0f);     //look-at position
//D3DXVECTOR3           m_vLookAtPt (0.0f, -70.0f, -30.0f);     //look-at position
D3DXVECTOR3             m_vUp (0.0f, 1.0f, 0.0f);                       //up direction

bool UpdateCamera (void) {

        RotateCamera();
        MoveCamera();

        return 1;
}

void RotateCamera (void) {

        staticint m_state = 0;
        static POINT m_pos;

        if(m_MouseState.rgbButtons[1] & 0x80)
        {
                //set mouse position
                if (m_state==0) {
                        m_state=1;
                        GetCursorPos(&m_pos);
                        ShowCursor(FALSE);
                }

                SetCursorPos(10000, 10000);


                D3DXVECTOR3 vDirection,vRotAxis;
                D3DXMATRIX matRotAxis;

                D3DXVec3Normalize(&vDirection,
                                                &(m_vLookAtPt - m_vEyePt)); //create direction vector

                if(m_KeyBuffer[DIK_LSHIFT] & 0x80) { //fast movement (shift)
                        //rotate Y
                        D3DXVec3Cross(&vRotAxis,&vDirection,&m_vUp); //rotation axis
```

```
                        //create rotation matrices
                        D3DXMatrixRotationAxis(&matRotAxis, &vRotAxis, m_MouseState.lY*SENSITIVITY*-1*SHIFT_ROT_MULTIP);
                        D3DXVec3TransformCoord(&vDirection,&vDirection,&(matRotAxis));//rotate direction

                        //rotate X
                        D3DXVec3Cross(&vRotAxis,&vDirection,&vRotAxis); //rotation axis
                        //create rotation matrices
                        D3DXMatrixRotationAxis(&matRotAxis, &vRotAxis, m_MouseState.lX*SENSITIVITY*-1*SHIFT_ROT_MULTIP);
                        D3DXVec3TransformCoord(&vDirection,&vDirection,&(matRotAxis));//rotate direction

                } elseif (m_KeyBuffer[DIK_LCONTROL] & 0x80) { //slow movement (ctrl)
                        //rotate Y
                        D3DXVec3Cross(&vRotAxis,&vDirection,&m_vUp); //rotation axis
                        //create rotation matrices
                        D3DXMatrixRotationAxis(&matRotAxis, &vRotAxis, m_MouseState.lY*SENSITIVITY*-1*CTRL_ROT_MULTIP);
                        D3DXVec3TransformCoord(&vDirection,&vDirection,&(matRotAxis));//rotate direction

                        //rotate X
                        D3DXVec3Cross(&vRotAxis,&vDirection,&vRotAxis); //rotation axis
                        //create rotation matrices
                        D3DXMatrixRotationAxis(&matRotAxis, &vRotAxis, m_MouseState.lX*SENSITIVITY*-1*CTRL_ROT_MULTIP);
                        D3DXVec3TransformCoord(&vDirection,&vDirection,&(matRotAxis));//rotate direction

                } else {
                        //rotate Y
                        D3DXVec3Cross(&vRotAxis,&vDirection,&m_vUp); //rotation axis
                        D3DXMatrixRotationAxis(&matRotAxis, &vRotAxis, m_MouseState.lY*SENSITIVITY*-1); //create rotation
matrices
                        D3DXVec3TransformCoord(&vDirection,&vDirection,&(matRotAxis));//rotate direction

                        //rotate X
                        D3DXVec3Cross(&vRotAxis,&vDirection,&vRotAxis); //rotation axis
                        D3DXMatrixRotationAxis(&matRotAxis, &vRotAxis, m_MouseState.lX*SENSITIVITY*-1); //create rotation
matrices
                        D3DXVec3TransformCoord(&vDirection,&vDirection,&(matRotAxis));//rotate direction
                }

                //translate up vector
                m_vLookAtPt = vDirection + m_vEyePt;

        } else {
                if (m_state==1) {
                        m_state = 0;
                        SetCursorPos((int) m_pos.x, (int) m_pos.y);
                        ShowCursor(TRUE);
                }

        }
}


void MoveCamera (void) {
        D3DXVECTOR3 vDirection;

        D3DXVec3Normalize(&vDirection, &(m_vLookAtPt - m_vEyePt)); //create direction vector

        //zoom in (W or wheel)
        if(m_MouseState.lZ > 0) //wheel rotated (priority)
        {
                m_vEyePt += vDirection * SPEED * WHEEL_MULTIP;
                m_vLookAtPt += vDirection * SPEED * WHEEL_MULTIP;
        } elseif (m_KeyBuffer[DIK_W] & 0x80) { //w pressed

                if(m_KeyBuffer[DIK_LSHIFT] & 0x80) { //fast movement (shift)
                        m_vEyePt += vDirection * SPEED * SHIFT_MOV_MULTIP;
                        m_vLookAtPt += vDirection * SPEED * SHIFT_MOV_MULTIP;
                } elseif (m_KeyBuffer[DIK_LCONTROL] & 0x80) { //slow movement (ctrl)
                        m_vEyePt += vDirection * SPEED * CTRL_MOV_MULTIP;
                        m_vLookAtPt += vDirection * SPEED * CTRL_MOV_MULTIP;
                } else {
                        m_vEyePt += vDirection * SPEED;
                        m_vLookAtPt += vDirection * SPEED;
                }

        }
```

```
//zoom out (S or wheel)
if(m_MouseState.lZ < 0) //wheel rotated (priority)
{
        m_vEyePt -= vDirection * SPEED * WHEEL_MULTIP;
        m_vLookAtPt -= vDirection * SPEED * WHEEL_MULTIP;
} elseif (m_KeyBuffer[DIK_S] & 0x80) { //s pressed

        if(m_KeyBuffer[DIK_LSHIFT] & 0x80) { //fast movement (shift)
                m_vEyePt -= vDirection * SPEED * SHIFT_MOV_MULTIP;
                m_vLookAtPt -= vDirection * SPEED * SHIFT_MOV_MULTIP;
        } elseif (m_KeyBuffer[DIK_LCONTROL] & 0x80) { //slow movement (ctrl)
                m_vEyePt -= vDirection * SPEED * CTRL_MOV_MULTIP;
                m_vLookAtPt -= vDirection * SPEED * CTRL_MOV_MULTIP;
        } else {
                m_vEyePt -= vDirection * SPEED;
                m_vLookAtPt -= vDirection * SPEED;
        }

}

//move left (A)
if(m_KeyBuffer[DIK_A] & 0x80)
{
        D3DXVec3Cross(&vDirection,&vDirection,&m_vUp);
        D3DXVec3Normalize(&vDirection,&vDirection);

        if(m_KeyBuffer[DIK_LSHIFT] & 0x80) { //fast movement (shift)
                m_vEyePt += vDirection * SPEED * SHIFT_MOV_MULTIP;
                m_vLookAtPt += vDirection * SPEED * SHIFT_MOV_MULTIP;
        } elseif (m_KeyBuffer[DIK_LCONTROL] & 0x80) { //slow movement (ctrl)
                m_vEyePt += vDirection * SPEED * CTRL_MOV_MULTIP;
                m_vLookAtPt += vDirection * SPEED * CTRL_MOV_MULTIP;
        } else {
                m_vEyePt += vDirection * SPEED;
                m_vLookAtPt += vDirection * SPEED;
        }
}

//move right (D)
if(m_KeyBuffer[DIK_D] & 0x80)
{
        D3DXVec3Cross(&vDirection,&vDirection,&m_vUp);
        D3DXVec3Normalize(&vDirection,&vDirection);

        if(m_KeyBuffer[DIK_LSHIFT] & 0x80) { //fast movement (shift)
                m_vEyePt -= vDirection * SPEED * SHIFT_MOV_MULTIP;
                m_vLookAtPt -= vDirection * SPEED * SHIFT_MOV_MULTIP;
        } elseif (m_KeyBuffer[DIK_LCONTROL] & 0x80) { //slow movement (ctrl)
                m_vEyePt -= vDirection * SPEED * CTRL_MOV_MULTIP;
                m_vLookAtPt -= vDirection * SPEED * CTRL_MOV_MULTIP;
        } else {
                m_vEyePt -= vDirection * SPEED;
                m_vLookAtPt -= vDirection * SPEED;
        }
}

//move up (R)
if(m_KeyBuffer[DIK_R] & 0x80)
{
        vDirection = m_vUp;
        D3DXVec3Normalize(&vDirection,&vDirection);

        if(m_KeyBuffer[DIK_LSHIFT] & 0x80) { //fast movement (shift)
                m_vEyePt += vDirection * SPEED * SHIFT_MOV_MULTIP;
                m_vLookAtPt += vDirection * SPEED * SHIFT_MOV_MULTIP;
        } elseif (m_KeyBuffer[DIK_LCONTROL] & 0x80) { //slow movement (ctrl)
                m_vEyePt += vDirection * SPEED * CTRL_MOV_MULTIP;
                m_vLookAtPt += vDirection * SPEED * CTRL_MOV_MULTIP;
        } else {
                m_vEyePt += vDirection * SPEED;
                m_vLookAtPt += vDirection * SPEED;
        }
}

//move down (F)
if(m_KeyBuffer[DIK_F] & 0x80)
```

```
{
        vDirection = m_vUp;
        D3DXVec3Normalize(&vDirection,&vDirection);

        if(m_KeyBuffer[DIK_LSHIFT] & 0x80) { //fast movement (shift)
                m_vEyePt -= vDirection * SPEED * SHIFT_MOV_MULTIP;
                m_vLookAtPt -= vDirection * SPEED * SHIFT_MOV_MULTIP;
        } elseif (m_KeyBuffer[DIK_LCONTROL] & 0x80) { //slow movement (ctrl)
                m_vEyePt -= vDirection * SPEED * CTRL_MOV_MULTIP;
                m_vLookAtPt -= vDirection * SPEED * CTRL_MOV_MULTIP;
        } else {
                m_vEyePt -= vDirection * SPEED;
                m_vLookAtPt -= vDirection * SPEED;
        }
}

D3DXVECTOR3 vRotAxis;
D3DXMATRIX matRotAxis;

D3DXVec3Normalize(&vDirection, &(m_vLookAtPt - m_vEyePt));

D3DXVec3Cross(&vRotAxis,&vDirection,&m_vUp);

//till up (T)
if(m_KeyBuffer[DIK_T] & 0x80)
{
        if(m_KeyBuffer[DIK_LSHIFT] & 0x80) { //fast movement (shift)
                D3DXMatrixRotationAxis(&matRotAxis, &vRotAxis, 5*SENSITIVITY*-1*SHIFT_ROT_MULTIP);
        } elseif (m_KeyBuffer[DIK_LCONTROL] & 0x80) { //slow movement (ctrl)
                D3DXMatrixRotationAxis(&matRotAxis, &vRotAxis, 5*SENSITIVITY*-1*CTRL_ROT_MULTIP);
        } else {
                D3DXMatrixRotationAxis(&matRotAxis, &vRotAxis, 5*SENSITIVITY*-1);
        }
        D3DXVec3TransformCoord(&vDirection,&vDirection,&(matRotAxis));//rotate direction
        m_vLookAtPt = vDirection + m_vEyePt;
}

//till down (G)
if(m_KeyBuffer[DIK_G] & 0x80)
{
        if(m_KeyBuffer[DIK_LSHIFT] & 0x80) { //fast movement (shift)
                D3DXMatrixRotationAxis(&matRotAxis, &vRotAxis, 5*SENSITIVITY*SHIFT_ROT_MULTIP);
        } elseif (m_KeyBuffer[DIK_LCONTROL] & 0x80) { //slow movement (ctrl)
                D3DXMatrixRotationAxis(&matRotAxis, &vRotAxis, 5*SENSITIVITY*CTRL_ROT_MULTIP);
        } else {
                D3DXMatrixRotationAxis(&matRotAxis, &vRotAxis, 5*SENSITIVITY);
        }
        D3DXVec3TransformCoord(&vDirection,&vDirection,&(matRotAxis));//rotate direction
        m_vLookAtPt = vDirection + m_vEyePt;
}

D3DXVec3Cross(&vRotAxis,&vDirection,&vRotAxis);

//rotate right (E)
if(m_KeyBuffer[DIK_E] & 0x80)
 {
        if(m_KeyBuffer[DIK_LSHIFT] & 0x80) { //fast movement (shift)
                D3DXMatrixRotationAxis(&matRotAxis, &vRotAxis, 5*SENSITIVITY*-1*SHIFT_ROT_MULTIP);
        } elseif (m_KeyBuffer[DIK_LCONTROL] & 0x80) { //slow movement (ctrl)
                D3DXMatrixRotationAxis(&matRotAxis, &vRotAxis, 5*SENSITIVITY*-1*CTRL_ROT_MULTIP);
        } else {
                D3DXMatrixRotationAxis(&matRotAxis, &vRotAxis, 5*SENSITIVITY*-1);
        }
        D3DXVec3TransformCoord(&vDirection,&vDirection,&(matRotAxis));//rotate direction
        m_vLookAtPt = vDirection + m_vEyePt;
}

//rotate left (Q)
if(m_KeyBuffer[DIK_Q] & 0x80)
 {

        if(m_KeyBuffer[DIK_LSHIFT] & 0x80) { //fast movement (shift)
                D3DXMatrixRotationAxis(&matRotAxis, &vRotAxis, 5*SENSITIVITY*SHIFT_ROT_MULTIP);
        } elseif (m_KeyBuffer[DIK_LCONTROL] & 0x80) { //slow movement (ctrl)
                D3DXMatrixRotationAxis(&matRotAxis, &vRotAxis, 5*SENSITIVITY*CTRL_ROT_MULTIP);
        } else {
```

```
                        D3DXMatrixRotationAxis(&matRotAxis, &vRotAxis, 5*SENSITIVITY);
                }
                D3DXVec3TransformCoord(&vDirection,&vDirection,&(matRotAxis));//rotate direction
                m_vLookAtPt = vDirection + m_vEyePt;
        }

}


#endif
```

5.5 input.h

```
#ifndef INPUT
#define INPUT

#include<dinput.h>

LPDIRECTINPUT8                  m_pDIObject = NULL;
LPDIRECTINPUTDEVICE8    m_pDIKeyboardDevice = NULL;
LPDIRECTINPUTDEVICE8    m_pDIMouseDevice = NULL;

char                    m_KeyBuffer[256];
DIMOUSESTATE2   m_MouseState;

bool LoadInput (HWND hWnd) {

        //General
        if (DirectInput8Create(GetModuleHandle(NULL), DIRECTINPUT_VERSION, IID_IDirectInput8A, (void**)&m_pDIObject, NULL)
!= DI_OK) { return 0;} //start direxTinput8

        //Keyboard

        //start keyboard device listening
        if (m_pDIObject->CreateDevice(GUID_SysKeyboard,&m_pDIKeyboardDevice,NULL) != DI_OK) { return 0;}
        //set format (for keyboard)
        if (m_pDIKeyboardDevice->SetDataFormat(&c_dfDIKeyboard) != DI_OK) { return 0;}
        //set cooperative level
        if (m_pDIKeyboardDevice->SetCooperativeLevel(hWnd, DISCL_FOREGROUND | DISCL_NONEXCLUSIVE) != DI_OK) { return 0;}
        //get access to the device
        if (m_pDIKeyboardDevice->Acquire() != DI_OK) {return 0;}

        //Mouse

        //start mouse device listening
        if (m_pDIObject->CreateDevice(GUID_SysMouse,&m_pDIMouseDevice,NULL) != DI_OK) { return 0;}
        //set format (for mouse)
        if (m_pDIMouseDevice->SetDataFormat(&c_dfDIMouse2) != DI_OK) { return 0;}
        //set cooperative level
        if (m_pDIMouseDevice->SetCooperativeLevel(hWnd, DISCL_FOREGROUND | DISCL_NONEXCLUSIVE) != DI_OK) { return 0;}
        //get access to the device
        if (m_pDIMouseDevice->Acquire() != DI_OK) {return 0;}


        return 1;
}

bool UpdateInput (void) {

        //read keyboard state
        if (m_pDIKeyboardDevice->GetDeviceState(sizeof(m_KeyBuffer),(LPVOID)&m_KeyBuffer) != DI_OK) { return 0;}
        //read mouse state
        if (m_pDIMouseDevice->GetDeviceState(sizeof(m_MouseState),(LPVOID)&m_MouseState) != DI_OK)  { return 0;}

        return 1;
}


#endif
```

5.6 console.h

```
#ifndef console
#define console

#include<d3d9.h>
#include<d3dx9.h>
#include<string>
#include<iostream>
#include<time.h>
#include<WinDef.h>
#include"global_defines.h"

class console_class {
public:
        bool state; //console is displayed or not
        bool done; //text submited or not
        char keydown; //is a key pressed
        std::string text; //text in the console input
        std::string comand; //submited text
        std::string output[3]; //response of server

        RECT rect1; //text field coordinates
        RECT rect2; //
        RECT rect3; //
        RECT rect4; //

        LPDIRECT3DVERTEXBUFFER9 buffer; //vertex buffer
        LPDIRECT3DTEXTURE9 texture;             //vertex texture

        VERTEX_2D vertices[6];  //6 vertice
};

console_class Console; //the class to the console

std::string KeyProc(int i, bool shift);
void ConsoleProc(void);
void ConsoleLoadVertices(void);
void ConsoleInit (LPDIRECT3DDEVICE9 d3ddev);

void ConsoleInit(LPDIRECT3DDEVICE9 d3ddev) {

        Console.state = 0; //console is displayed or not
        Console.done = 0; //text submited or not
        Console.keydown = 0; //is a key pressed
        Console.text = ""; //text in the console input
        Console.comand = "";

        Console.output[0] = ""; //response of programm
        Console.output[1] = ""; //response of programm
        Console.output[2] = ""; //response of programm

        //text fileds
        Console.rect1.left = 10;
        Console.rect1.top = SCREEN_HEIGHT-125;
        Console.rect1.right = SCREEN_WIDTH-20;
        Console.rect1.bottom = SCREEN_HEIGHT-100;

        Console.rect2.left = 10;
        Console.rect2.top = SCREEN_HEIGHT-100;
        Console.rect2.right = SCREEN_WIDTH-20;
        Console.rect2.bottom = SCREEN_HEIGHT-75;

        Console.rect3.left = 10;
        Console.rect3.top = SCREEN_HEIGHT-75;
        Console.rect3.right = SCREEN_WIDTH-20;
        Console.rect3.bottom = SCREEN_HEIGHT-50;
        Console.rect4.left = 20;
        Console.rect4.top = SCREEN_HEIGHT-35;
        Console.rect4.right = SCREEN_WIDTH-20;
        Console.rect4.bottom = SCREEN_HEIGHT-10;
```

```
	ConsoleLoadVertices();

	d3ddev->CreateVertexBuffer(6*sizeof(VERTEX_2D), 0, FVF_2D, D3DPOOL_MANAGED, &Console.buffer, NULL);

	VOID* pVoid;
	Console.buffer->Lock(0, 0, (void**)&pVoid, 0);
	memcpy(pVoid, Console.vertices, 6*sizeof(VERTEX_2D));
	Console.buffer->Unlock();

	D3DXCreateTextureFromFileA(d3ddev, "GUI\\Image\\tConsole.png", &Console.texture);
}


void ConsoleLoadVertices(void){

	Console.vertices[0].X = 0.0f;                                    Console.vertices[0].Y = (float) SCREEN_HEIGHT;
	Console.vertices[0].Z = 0.5f; Console.vertices[0].RHW= 1.0f; Console.vertices[0].U = 0.1;Console.vertices[0].V = 1;
	Console.vertices[1].X = (float) SCREEN_WIDTH;                    Console.vertices[1].Y = (float) SCREEN_HEIGHT;
	Console.vertices[1].Z = 0.5f; Console.vertices[1].RHW= 1.0f; Console.vertices[1].U = 1; Console.vertices[1].V = 1;
	Console.vertices[2].X = 0.0f;                                    Console.vertices[2].Y = (float) SCREEN_HEIGHT-125;
	Console.vertices[2].Z = 0.5f; Console.vertices[2].RHW= 1.0f; Console.vertices[2].U = 0.1;Console.vertices[2].V = 0;

	Console.vertices[3].X = (float) SCREEN_WIDTH;                    Console.vertices[3].Y = (float) SCREEN_HEIGHT;
	Console.vertices[3].Z = 0.5f; Console.vertices[3].RHW= 1.0f; Console.vertices[3].U = 1; Console.vertices[3].V = 1;
	Console.vertices[4].X = (float) SCREEN_WIDTH;                    Console.vertices[4].Y = (float) SCREEN_HEIGHT-125;
	Console.vertices[4].Z = 0.5f; Console.vertices[4].RHW= 1.0f; Console.vertices[4].U = 1; Console.vertices[4].V = 0;
	Console.vertices[5].X = 0.0f;                                    Console.vertices[5].Y = (float) SCREEN_HEIGHT-125;
	Console.vertices[5].Z = 0.5f; Console.vertices[5].RHW= 1.0f; Console.vertices[5].U = 0.1;Console.vertices[5].V = 0;

}


void ConsoleProc(void) {
	bool shift = 0; //shift is pressed
	bool is_pressed = 0; //key is pressed
	static clock_t BS = clock(); //time backspace was pressed last
	staticint BS_count = 0; //times backspace was pressed

	shift = (m_KeyBuffer[DIK_LSHIFT] & 0x80 || m_KeyBuffer[DIK_LSHIFT] & 0x80); //is shift pressed

	is_pressed = 0;
	for (int i=0x00; i<=0xff; i+=0x01){ //check if a key is pressed
		if ((m_KeyBuffer[i] & 0x80) && i != DIK_RSHIFT && i != DIK_LSHIFT ) { //key is pressed (not shift)
			Console.text += KeyProc(i, shift); //procces the pressed key
			is_pressed = 1;
			break;
		}
	}

	if (is_pressed == 0) { Console.keydown = 0;} // set keydown to 0 if no key was pressed


	if (m_KeyBuffer[DIK_BACK] & 0x80 && Console.text != "") { //backspace if BS is pressed and input not empty
		if (clock()-BS > 150 || BS_count >= 3) {
			Console.text.erase (Console.text.length()-1);
			BS = clock();
			BS_count +=1;
		}
	} else {
		BS_count = 0;
	}

	if (m_KeyBuffer[DIK_RETURN] & 0x80 && Console.text != "") {
		Console.comand = Console.text;
		Console.text = "";
	}

}


std::string KeyProc(int i, bool shift) {

	std::string ret;
	int temp_keydown = 0;

	if (shift == 1) {
```

```
        if (i == DIK_A) { ret = "A"; temp_keydown = DIK_A;}
        elseif (i == DIK_B) { ret = "B"; temp_keydown = DIK_B;}
        elseif (i == DIK_C) { ret = "C"; temp_keydown = DIK_C;}
        elseif (i == DIK_D) { ret = "D"; temp_keydown = DIK_D;}
        elseif (i == DIK_E) { ret = "E"; temp_keydown = DIK_E;}
        elseif (i == DIK_F) { ret = "F"; temp_keydown = DIK_F;}
        elseif (i == DIK_G) { ret = "G"; temp_keydown = DIK_G;}
        elseif (i == DIK_H) { ret = "H"; temp_keydown = DIK_H;}
        elseif (i == DIK_I) { ret = "I"; temp_keydown = DIK_I;}
        elseif (i == DIK_J) { ret = "J"; temp_keydown = DIK_J;}
        elseif (i == DIK_K) { ret = "K"; temp_keydown = DIK_K;}
        elseif (i == DIK_L) { ret = "L"; temp_keydown = DIK_L;}
        elseif (i == DIK_M) { ret = "M"; temp_keydown = DIK_M;}
        elseif (i == DIK_N) { ret = "N"; temp_keydown = DIK_N;}
        elseif (i == DIK_O) { ret = "O"; temp_keydown = DIK_O;}
        elseif (i == DIK_P) { ret = "P"; temp_keydown = DIK_P;}
        elseif (i == DIK_Q) { ret = "Q"; temp_keydown = DIK_Q;}
        elseif (i == DIK_R) { ret = "R"; temp_keydown = DIK_R;}
        elseif (i == DIK_S) { ret = "S"; temp_keydown = DIK_S;}
        elseif (i == DIK_T) { ret = "T"; temp_keydown = DIK_T;}
        elseif (i == DIK_U) { ret = "U"; temp_keydown = DIK_U;}
        elseif (i == DIK_V) { ret = "V"; temp_keydown = DIK_V;}
        elseif (i == DIK_W) { ret = "W"; temp_keydown = DIK_W;}
        elseif (i == DIK_X) { ret = "X"; temp_keydown = DIK_X;}
        elseif (i == DIK_Y) { ret = "Z"; temp_keydown = DIK_Y;} //english keyboard
        elseif (i == DIK_Z) { ret = "Y"; temp_keydown = DIK_Z;} //english keyboard
        elseif (i == DIK_0) { ret = "="; temp_keydown = DIK_0;}
        elseif (i == DIK_1) { ret = "+"; temp_keydown = DIK_1;}
        elseif (i == DIK_2) { ret = "\""; temp_keydown = DIK_2;}
        elseif (i == DIK_3) { ret = "*"; temp_keydown = DIK_3;}
        elseif (i == DIK_4) { ret = "ç"; temp_keydown = DIK_4;}
        elseif (i == DIK_5) { ret = "%"; temp_keydown = DIK_5;}
        elseif (i == DIK_6) { ret = "&"; temp_keydown = DIK_6;}
        elseif (i == DIK_7) { ret = "/"; temp_keydown = DIK_7;}
        elseif (i == DIK_8) { ret = "("; temp_keydown = DIK_8;}
        elseif (i == DIK_9) { ret = ")"; temp_keydown = DIK_9;}
        elseif (i == DIK_SLASH) { ret = "_"; temp_keydown = DIK_SLASH;} //english keyboard
} else {
        if (i == DIK_A) { ret = "a"; temp_keydown = DIK_A;}
        elseif (i == DIK_B) { ret = "b"; temp_keydown = DIK_B;}
        elseif (i == DIK_C) { ret = "c"; temp_keydown = DIK_C;}
        elseif (i == DIK_D) { ret = "d"; temp_keydown = DIK_D;}
        elseif (i == DIK_E) { ret = "e"; temp_keydown = DIK_E;}
        elseif (i == DIK_F) { ret = "f"; temp_keydown = DIK_F;}
        elseif (i == DIK_G) { ret = "g"; temp_keydown = DIK_G;}
        elseif (i == DIK_H) { ret = "h"; temp_keydown = DIK_H;}
        elseif (i == DIK_I) { ret = "i"; temp_keydown = DIK_I;}
        elseif (i == DIK_J) { ret = "j"; temp_keydown = DIK_J;}
        elseif (i == DIK_K) { ret = "k"; temp_keydown = DIK_K;}
        elseif (i == DIK_L) { ret = "l"; temp_keydown = DIK_L;}
        elseif (i == DIK_M) { ret = "m"; temp_keydown = DIK_M;}
        elseif (i == DIK_N) { ret = "n"; temp_keydown = DIK_N;}
        elseif (i == DIK_O) { ret = "o"; temp_keydown = DIK_O;}
        elseif (i == DIK_P) { ret = "p"; temp_keydown = DIK_P;}
        elseif (i == DIK_Q) { ret = "q"; temp_keydown = DIK_Q;}
        elseif (i == DIK_R) { ret = "r"; temp_keydown = DIK_R;}
        elseif (i == DIK_S) { ret = "s"; temp_keydown = DIK_S;}
        elseif (i == DIK_T) { ret = "t"; temp_keydown = DIK_T;}
        elseif (i == DIK_U) { ret = "u"; temp_keydown = DIK_U;}
        elseif (i == DIK_V) { ret = "v"; temp_keydown = DIK_V;}
        elseif (i == DIK_W) { ret = "w"; temp_keydown = DIK_W;}
        elseif (i == DIK_X) { ret = "x"; temp_keydown = DIK_X;}
        elseif (i == DIK_Y) { ret = "z"; temp_keydown = DIK_Y;} //english keyboard
        elseif (i == DIK_Z) { ret = "y"; temp_keydown = DIK_Z;} //english keyboard
        elseif (i == DIK_0) { ret = "0"; temp_keydown = DIK_0;}
        elseif (i == DIK_1) { ret = "1"; temp_keydown = DIK_1;}
        elseif (i == DIK_2) { ret = "2"; temp_keydown = DIK_2;}
        elseif (i == DIK_3) { ret = "3"; temp_keydown = DIK_3;}
        elseif (i == DIK_4) { ret = "4"; temp_keydown = DIK_4;}
        elseif (i == DIK_5) { ret = "5"; temp_keydown = DIK_5;}
        elseif (i == DIK_6) { ret = "6"; temp_keydown = DIK_6;}
        elseif (i == DIK_7) { ret = "7"; temp_keydown = DIK_7;}
        elseif (i == DIK_8) { ret = "8"; temp_keydown = DIK_8;}
        elseif (i == DIK_9) { ret = "9"; temp_keydown = DIK_9;}
        elseif (i == DIK_PERIOD) { ret = "."; temp_keydown = DIK_PERIOD;}
        elseif (i == DIK_COMMA) { ret = ","; temp_keydown = DIK_COMMA;}
```

```
                elseif (i == DIK_SLASH) { ret = "-"; temp_keydown = DIK_SLASH;} //english keyboard
        }

        if (i == DIK_SPACE) { ret = " "; temp_keydown = DIK_SPACE;}

        if (temp_keydown == Console.keydown) {
                ret = "";
        } else {
                Console.keydown = temp_keydown;
        }

        return ret;

}


#endif
```

5.7 info.h

```
#ifndef info
#define info

#include<WinDef.h>
#include"color_defines.h"

class info_class {
public:
        RECT line1;
        RECT line2;
        RECT line3;
        RECT line4;

};

info_class Info;

voidInfoInit(void);

voidInfoInit(void) {
        Info.line1.left = 10; Info.line1.top = 10; Info.line1.right = SCREEN_WIDTH-20; Info.line1.bottom = 35;
        Info.line2.left = 10; Info.line2.top = 35; Info.line2.right = SCREEN_WIDTH-20; Info.line2.bottom = 60;
        Info.line3.left = 10; Info.line3.top = 60; Info.line3.right = SCREEN_WIDTH-20; Info.line3.bottom = 85;
        Info.line4.left = 10; Info.line4.top = 85; Info.line4.right = SCREEN_WIDTH-20; Info.line4.bottom = 110;
}


#endif
```

5.8 form.h

```
#ifndef FORM
#define FORM

//includes
#include"HMap.h"
#include<d3d9.h>
#include<d3dx9.h>
#include<math.h>

//classes
struct VERTEX_FORM {FLOAT X, Y, Z;};
#define FVF_FORM (D3DFVF_XYZ)

class form_class {
public:
        int size;
        LPD3DXMESH mesh;// mesh pointer
```

```
        D3DMATERIAL9* material;    // define the material object
        LPDIRECT3DTEXTURE9* texture;    // a pointer to a texture
        DWORD numMaterials;    // number of naterials in the mesh

        VERTEX_FORM* vertices; //vertex information
        D3DXVECTOR3 speed; //vector whit actual speed and direction
        float RotSpeed; //holding actual rotation degree
        D3DXVECTOR3 RotAx; //vector holding actual rotation axis

        bool impact; //if last frame=impact

        float mass; //mass
        float x,y,z; //position
        float RotDeg; //rotation degree
};

class form_class2 { //class containing all forms (1 exists)
public:
        form_class Forms[200]; //array whit all forms
        int count; //number of active forms
        bool state; //move forms? (Y/N)
        D3DXVECTOR3 g; //gravitation
        float k;//"Restitutionskoeffizient"
        float l;
        float t_mult;   //time speed egg. 2x
};

//protoypes
void FormInit(void);
void FormCreate(LPDIRECT3DDEVICE9 d3ddev, float x, float y, float z);
void FormsCreate(LPDIRECT3DDEVICE9 d3ddev);
void FormsReset(LPDIRECT3DDEVICE9 d3ddev);
void FormUpdate(LPDIRECT3DDEVICE9 d3ddev);

int _Form_vtxListfromCoord(int x, int y);
float _Form_vtxHeightFromCoord(float x, float y);

void _Form_GroundImpact(form_class &form);


//global variables
form_class2 Forms;

void FormInit(void) { //initialize forms
        Forms.count = 0; //set forms count to 0
        //set global forces
        Forms.g = D3DXVECTOR3(0.0f, -9.81f, 0.0f); //gravitation
}


void FormCreate(LPDIRECT3DDEVICE9 d3ddev, float x, float y, float z) {

        LPDIRECT3DVERTEXBUFFER9 vb;             //vertex buffer (holds converted mesh data)
        D3DVERTEXBUFFER_DESC vbDesc;    //struct used to convert
        LPD3DXMESH mesh;                        //converted mesh
        VERTEX_FORM* vertices;                  //raw vertices

        LPD3DXBUFFER bufMaterial;
        char name[] = "stone";
        char buffer[255];

        sprintf_s(buffer, "Models\\%s\\%s.x", name, name);
        D3DXLoadMeshFromX(buffer,    // load this file
                        D3DXMESH_SYSTEMMEM,    // load the mesh into system memory
                        d3ddev,    // the Direct3D Device
                        NULL,    // we aren't using adjacency
&bufMaterial,    // put the materials here
                        NULL,    // we aren't using effect instances
                &Forms.Forms[Forms.count].numMaterials,    // the number of materials in this model
&Forms.Forms[Forms.count].mesh);    // put the mesh here

        // retrieve the pointer to the buffer containing the material information
        D3DXMATERIAL* tempMaterials = (D3DXMATERIAL*)bufMaterial->GetBufferPointer();

        // create a new material buffer and texture for each material in the mesh
        Forms.Forms[Forms.count].material = new D3DMATERIAL9[Forms.Forms[Forms.count].numMaterials];
```

```
	Forms.Forms[Forms.count].texture = new LPDIRECT3DTEXTURE9[Forms.Forms[Forms.count].numMaterials];

	for(DWORD i = 0; i < Forms.Forms[Forms.count].numMaterials; i++)	// for each material...
	{
		// get the material info
		Forms.Forms[Forms.count].material[i] = tempMaterials[i].MatD3D;
		// make ambient the same as diffuse
		Forms.Forms[Forms.count].material[i].Ambient = Forms.Forms[Forms.count].material[i].Diffuse;
		sprintf_s(buffer, "Models\\%s\\%s", name, tempMaterials[i].pTextureFilename);
		// if there is no texture, set the texture to NULL
		if(D3DXCreateTextureFromFileA(d3ddev, buffer, &Forms.Forms[Forms.count].texture[i]) != D3D_OK) {
			D3DXCreateTextureFromFileA(d3ddev, "Models\\NULL.png", &Forms.Forms[Forms.count].texture[i]);
		}
	}

	Forms.Forms[Forms.count].mesh->CloneMeshFVF(NULL, FVF_FORM, d3ddev, &mesh); //copy mesh and change FVF

	mesh->GetVertexBuffer(&vb);	//get meshs vertex buffer
	vb->GetDesc(&vbDesc);		//get information about vertex buffer
	vb->Lock(0,vbDesc.Size,(void**)&vertices,0); //lock vertex buffer and copy data into vertices

	int pos = 0; //actual count
	bool multiple; //is multiple occurence
	for(unsignedint i=0; i < vbDesc.Size/sizeof(VERTEX_FORM); i++){ //for all vertices
		multiple = false;
		for (unsignedint j=0; j < i; j++) { //for all already controlled vertices
			if( vertices[i].X == vertices[j].X &&//if are the same
				vertices[i].Y == vertices[j].Y &&
				vertices[i].Z == vertices[j].Z) {
				multiple = true;
			}
		}

		if(!(multiple)) { //if no multiple occourence
			vertices[pos] = vertices[i];
			pos += 1;
		}
	}

	Forms.Forms[Forms.count].size = pos; //ammount of vertices
	//free space in Form for vertices
	Forms.Forms[Forms.count].vertices = new VERTEX_FORM[Forms.Forms[Forms.count].size];

	for (int i=0; i<Forms.Forms[Forms.count].size;i++){ //copy vertices into Form
		Forms.Forms[Forms.count].vertices[i].X = vertices[i].X;
		Forms.Forms[Forms.count].vertices[i].Y = vertices[i].Y;
		Forms.Forms[Forms.count].vertices[i].Z = vertices[i].Z;
	}

	//init position
	Forms.Forms[Forms.count].x = x;
	Forms.Forms[Forms.count].y = y;
	Forms.Forms[Forms.count].z = z;
	//init speed
	Forms.Forms[Forms.count].speed.x = 0.0f;
	Forms.Forms[Forms.count].speed.y = 0.0f;
	Forms.Forms[Forms.count].speed.z = 0.0f;
	//init rotation
	Forms.Forms[Forms.count].RotAx.x = 0.0f;
	Forms.Forms[Forms.count].RotAx.z = 0.0f;
	Forms.Forms[Forms.count].RotAx.y = 0.0f;
	Forms.Forms[Forms.count].RotDeg = 0.0f;
	Forms.Forms[Forms.count].RotSpeed = 0.0f;

	Forms.Forms[Forms.count].impact = 0;

	Forms.Forms[Forms.count].mass = 1000;

	Forms.count += 1;
}



void FormUpdate(LPDIRECT3DDEVICE9 d3ddev) {
```

```
	for(int i=0; i<Forms.count; i++) {
		Forms.Forms[i].speed += Forms.g/(FPS[0]);///Forms.t_mult;

		Forms.Forms[i].x += Forms.Forms[i].speed.x/(FPS[0])*Forms.t_mult;
		Forms.Forms[i].y += Forms.Forms[i].speed.y/(FPS[0])*Forms.t_mult;
		Forms.Forms[i].z += Forms.Forms[i].speed.z/(FPS[0])*Forms.t_mult;

		Forms.Forms[i].RotDeg += Forms.Forms[i].RotSpeed/(FPS[0])*Forms.t_mult;
		if(Forms.Forms[i].RotDeg > 2*PI) { Forms.Forms[i].RotDeg = Forms.Forms[i].RotDeg-(2*PI); }

		_Form_GroundImpact(Forms.Forms[i]);
	}

}


void _Form_GroundImpact(form_class &form) {

	//Incomplete Documentation!
	//see phyik.docx for further information

	D3DXVECTOR3 r1; // position vector of ground impact triangle
	D3DXVECTOR3 r2; //
	D3DXVECTOR3 r3; //

	D3DXVECTOR3 v1; //r1-r3
	D3DXVECTOR3 v2; //r2-r3

	D3DXVECTOR3 vnormal, vparall; //speed normal & parallel to ground
	D3DXVECTOR3 fnormal, fparall; //force normal & parallel to ground
	D3DXVECTOR3 friction;

	D3DXVECTOR3 rform; //position vector of form barycenter
	D3DXMATRIX matRotAxis; //rotation matrix
	D3DXMatrixRotationAxis(&matRotAxis, &form.RotAx, form.RotDeg); //create rotation matrix

	float x, z, height, u, v;
	float height_cng = 0, last_height_cng = 0;


	for(int i=0; i<form.size; i++){ //for each form edge

		rform.x = form.vertices[i].X; //position vector of form edge in object coordinats
		rform.y = form.vertices[i].Y; //
		rform.z = form.vertices[i].Z; //

		D3DXVec3TransformCoord(&rform, &rform, &matRotAxis); //rotate form edge

		x=form.x+ rform.x; //global x coodrinate of form edge
		z=form.z+ rform.z; //global y coodrinate of form edge

		//get position vector of ground impact triangle
		r1.x =  HMap.vertices[_Form_vtxListfromCoord((int) floor(x), (int) floor(z+1))].X;
		r1.y =  HMap.vertices[_Form_vtxListfromCoord((int) floor(x), (int) floor(z+1))].Y;
		r1.z =  HMap.vertices[_Form_vtxListfromCoord((int) floor(x), (int) floor(z+1))].Z;

		r2.x =  HMap.vertices[_Form_vtxListfromCoord((int) floor(x+1), (int) floor(z))].X;
		r2.y =  HMap.vertices[_Form_vtxListfromCoord((int) floor(x+1), (int) floor(z))].Y;
		r2.z =  HMap.vertices[_Form_vtxListfromCoord((int) floor(x+1), (int) floor(z))].Z;

		if ((x-floor(x))+(z-floor(z)) < 1) {
			r3.x =  HMap.vertices[_Form_vtxListfromCoord((int) floor(x), (int) floor(z))].X;
			r3.y =  HMap.vertices[_Form_vtxListfromCoord((int) floor(x), (int) floor(z))].Y;
			r3.z =  HMap.vertices[_Form_vtxListfromCoord((int) floor(x), (int) floor(z))].Z;

			v1 = r1-r3;
			v2 = r2-r3;

			u =    (v2.z*(x-floor(x))-v2.x*(z-floor(z)))/(v1.x*v2.z - v1.z*v2.x);
			v = -1*(v1.z*(x-floor(x))-v1.x*(z-floor(z)))/(v1.x*v2.z - v1.z*v2.x);
			height = v1.y*u+v2.y*v+r3.y;

		} else {
			r3.x =  HMap.vertices[_Form_vtxListfromCoord((int) floor(x+1), (int) floor(z+1))].X;
			r3.y =  HMap.vertices[_Form_vtxListfromCoord((int) floor(x+1), (int) floor(z+1))].Y;
			r3.z =  HMap.vertices[_Form_vtxListfromCoord((int) floor(x+1), (int) floor(z+1))].Z;
```

```
                        v1 = r1-r3;
                        v2 = r2-r3;

                        u =    (v2.z*(x-floor(x+1))-v2.x*(z-floor(z+1)))/(v1.x*v2.z - v1.z*v2.x);
                        v = -1*(v1.z*(x-floor(x+1))-v1.x*(z-floor(z+1)))/(v1.x*v2.z - v1.z*v2.x);
                        height = v1.y*u+v2.y*v+r3.y;
                }

                if(height>=form.y+ rform.y) { //if edge touchs ground
                        height_cng = height-rform.y-form.y;
                        if (height_cng > last_height_cng) {
                                last_height_cng = height_cng;
                                D3DXVec3Cross(&vnormal, &v1, &v2);
                                D3DXVec3Cross(&fnormal, &v1, &v2);
                                vnormal = -D3DXVec3Dot(&form.speed,
                                &vnormal)/(D3DXVec3Length(&vnormal)*D3DXVec3Length(&vnormal))*vnormal;
                                fnormal = -D3DXVec3Dot(&(Forms.g*form.mass),
                                &fnormal)/(D3DXVec3Length(&fnormal)*D3DXVec3Length(&fnormal))*fnormal;
                                vparall = form.speed+vnormal;
                                fparall = (Forms.g*form.mass)+fnormal;
                        }
                }
        }



        if (last_height_cng != 0) { //if touch ground
                form.y += last_height_cng;

                if (form.impact == false) { //if impact (last frame = no impact)
                        form.speed = (Forms.k+1)*vnormal+form.speed;

                } else {//if slide (last frame = impact)

                        D3DXVec3Normalize(&friction, &vparall);
                        friction = -friction * (D3DXVec3Length(&fnormal)*Forms.l);
                        fparall += friction;

                        form.speed = vparall+(fparall/form.mass/(FPS[0])*Forms.t_mult);

                }
                form.impact = true;

        } else {
                form.impact = false;
        }


}

int _Form_vtxListfromCoord (int x, int y) { //get list ID in HMAP.vertices[id] from x, y coord (int!)
        return ((y-1)*(HMAP_WIDTH)+(HMAP_WIDTH-x)-1);
}

float _Form_vtxHeightFromCoord(float x, float y) { //get H-Map height from x, y coords

        D3DXVECTOR3 r1;
        D3DXVECTOR3 r2;
        D3DXVECTOR3 r3;

        D3DXVECTOR3 v1;
        D3DXVECTOR3 v2;

        float height, u, v;

        r1.x =  HMap.vertices[_Form_vtxListfromCoord((int) floor(x), (int) floor(y+1))].X;
        r1.y =  HMap.vertices[_Form_vtxListfromCoord((int) floor(x), (int) floor(y+1))].Y;
        r1.z =  HMap.vertices[_Form_vtxListfromCoord((int) floor(x), (int) floor(y+1))].Z;

        r2.x =  HMap.vertices[_Form_vtxListfromCoord((int) floor(x+1), (int) floor(y))].X;
        r2.y =  HMap.vertices[_Form_vtxListfromCoord((int) floor(x+1), (int) floor(y))].Y;
        r2.z =  HMap.vertices[_Form_vtxListfromCoord((int) floor(x+1), (int) floor(y))].Z;

        if ((x-floor(x))+(y-floor(y)) < 1) {
```

```
                r3.x =  HMap.vertices[_Form_vtxListfromCoord((int) floor(x), (int) floor(y))].X;
                r3.y =  HMap.vertices[_Form_vtxListfromCoord((int) floor(x), (int) floor(y))].Y;
                r3.z =  HMap.vertices[_Form_vtxListfromCoord((int) floor(x), (int) floor(y))].Z;

                v1 = r1-r3;
                v2 = r2-r3;

                u =    (v2.z*(x-floor(x))-v2.x*(y-floor(y)))/(v1.x*v2.z - v1.z*v2.x);
                v = -1*(v1.z*(x-floor(x))-v1.x*(y-floor(y)))/(v1.x*v2.z - v1.z*v2.x);
                height = v1.y*u+v2.y*v+r3.y;

        } else {
                r3.x =  HMap.vertices[_Form_vtxListfromCoord((int) floor(x+1), (int) floor(y+1))].X;
                r3.y =  HMap.vertices[_Form_vtxListfromCoord((int) floor(x+1), (int) floor(y+1))].Y;
                r3.z =  HMap.vertices[_Form_vtxListfromCoord((int) floor(x+1), (int) floor(y+1))].Z;

                v1 = r1-r3;
                v2 = r2-r3;

                u =    (v2.z*(x-floor(x+1))-v2.x*(y-floor(y+1)))/(v1.x*v2.z - v1.z*v2.x);
                v = -1*(v1.z*(x-floor(x+1))-v1.x*(y-floor(y+1)))/(v1.x*v2.z - v1.z*v2.x);
                height = v1.y*u+v2.y*v+r3.y;
        }

        return height;
}

void FormsReset(LPDIRECT3DDEVICE9 d3ddev) { //reset all forms to original position and speed
        Forms.count = 0;
        FormsCreate(d3ddev);
}

void FormsCreate(LPDIRECT3DDEVICE9 d3ddev) {    //create forms (those: sciora landslide! replace whit file?)

        FormCreate(d3ddev, 435.5, _Form_vtxHeightFromCoord(435.5, 180.0)+2, 180.0);
        FormCreate(d3ddev, 435.0, _Form_vtxHeightFromCoord(435.0, 180.0)+2, 180.0);
        FormCreate(d3ddev, 434.5, _Form_vtxHeightFromCoord(434.5, 180.0)+2, 180.0);
        FormCreate(d3ddev, 434.0, _Form_vtxHeightFromCoord(434.0, 180.0)+2, 180.0);
        FormCreate(d3ddev, 433.5, _Form_vtxHeightFromCoord(433.5, 180.0)+2, 180.0);
        FormCreate(d3ddev, 433.0, _Form_vtxHeightFromCoord(433.0, 180.0)+2, 180.0);
        FormCreate(d3ddev, 432.5, _Form_vtxHeightFromCoord(432.5, 180.0)+2, 180.0);
        FormCreate(d3ddev, 432.0, _Form_vtxHeightFromCoord(432.0, 180.0)+2, 180.0);

        FormCreate(d3ddev, 435.5, _Form_vtxHeightFromCoord(435.5, 179.5)+2, 179.5);
        FormCreate(d3ddev, 435.0, _Form_vtxHeightFromCoord(435.0, 179.5)+2, 179.5);
        FormCreate(d3ddev, 434.5, _Form_vtxHeightFromCoord(434.5, 179.5)+2, 179.5);
        FormCreate(d3ddev, 434.0, _Form_vtxHeightFromCoord(434.0, 179.5)+2, 179.5);
        FormCreate(d3ddev, 433.5, _Form_vtxHeightFromCoord(433.5, 179.5)+2, 179.5);
        FormCreate(d3ddev, 433.0, _Form_vtxHeightFromCoord(433.0, 179.5)+2, 179.5);
        FormCreate(d3ddev, 432.5, _Form_vtxHeightFromCoord(432.5, 179.5)+2, 179.5);
        FormCreate(d3ddev, 432.0, _Form_vtxHeightFromCoord(432.0, 179.5)+2, 179.5);

        FormCreate(d3ddev, 436.0, _Form_vtxHeightFromCoord(436.0, 179.0)+2, 179.0);
        FormCreate(d3ddev, 435.5, _Form_vtxHeightFromCoord(435.5, 179.0)+2, 179.0);
        FormCreate(d3ddev, 435.0, _Form_vtxHeightFromCoord(435.0, 179.0)+2, 179.0);
        FormCreate(d3ddev, 434.5, _Form_vtxHeightFromCoord(434.5, 179.0)+2, 179.0);
        FormCreate(d3ddev, 434.0, _Form_vtxHeightFromCoord(434.0, 179.0)+2, 179.0);
        FormCreate(d3ddev, 433.5, _Form_vtxHeightFromCoord(433.5, 179.0)+2, 179.0);
        FormCreate(d3ddev, 433.0, _Form_vtxHeightFromCoord(433.0, 179.0)+2, 179.0);
        FormCreate(d3ddev, 432.5, _Form_vtxHeightFromCoord(432.5, 179.0)+2, 179.0);
        FormCreate(d3ddev, 432.0, _Form_vtxHeightFromCoord(432.0, 179.0)+2, 179.0);

        FormCreate(d3ddev, 436.5, _Form_vtxHeightFromCoord(436.5, 178.5)+2, 178.5);
        FormCreate(d3ddev, 436.0, _Form_vtxHeightFromCoord(436.0, 178.5)+2, 178.5);
        FormCreate(d3ddev, 435.5, _Form_vtxHeightFromCoord(435.5, 178.5)+2, 178.5);
        FormCreate(d3ddev, 435.0, _Form_vtxHeightFromCoord(435.0, 178.5)+2, 178.5);
        FormCreate(d3ddev, 434.5, _Form_vtxHeightFromCoord(434.5, 178.5)+2, 178.5);
        FormCreate(d3ddev, 434.0, _Form_vtxHeightFromCoord(434.0, 178.5)+2, 178.5);
        FormCreate(d3ddev, 433.5, _Form_vtxHeightFromCoord(433.5, 178.5)+2, 178.5);
        FormCreate(d3ddev, 433.0, _Form_vtxHeightFromCoord(433.0, 178.5)+2, 178.5);
        FormCreate(d3ddev, 432.5, _Form_vtxHeightFromCoord(432.5, 178.5)+2, 178.5);
        FormCreate(d3ddev, 432.0, _Form_vtxHeightFromCoord(432.0, 178.5)+2, 178.5);
        FormCreate(d3ddev, 431.5, _Form_vtxHeightFromCoord(431.5, 178.5)+2, 178.5);

        FormCreate(d3ddev, 436.5, _Form_vtxHeightFromCoord(436.5, 178.0)+2, 178.0);
        FormCreate(d3ddev, 436.0, _Form_vtxHeightFromCoord(436.0, 178.0)+2, 178.0);
```

```
FormCreate(d3ddev, 435.5, _Form_vtxHeightFromCoord(435.5, 178.0)+2, 178.0);
FormCreate(d3ddev, 435.0, _Form_vtxHeightFromCoord(435.0, 178.0)+2, 178.0);
FormCreate(d3ddev, 434.5, _Form_vtxHeightFromCoord(434.5, 178.0)+2, 178.0);
FormCreate(d3ddev, 434.0, _Form_vtxHeightFromCoord(434.0, 178.0)+2, 178.0);
FormCreate(d3ddev, 433.5, _Form_vtxHeightFromCoord(433.5, 178.0)+2, 178.0);
FormCreate(d3ddev, 433.0, _Form_vtxHeightFromCoord(433.0, 178.0)+2, 178.0);
FormCreate(d3ddev, 432.5, _Form_vtxHeightFromCoord(432.5, 178.0)+2, 178.0);
FormCreate(d3ddev, 432.0, _Form_vtxHeightFromCoord(432.0, 178.0)+2, 178.0);
FormCreate(d3ddev, 431.5, _Form_vtxHeightFromCoord(431.5, 178.0)+2, 178.0);

FormCreate(d3ddev, 436.5, _Form_vtxHeightFromCoord(436.5, 177.5)+2, 177.5);
FormCreate(d3ddev, 436.0, _Form_vtxHeightFromCoord(436.0, 177.5)+2, 177.5);
FormCreate(d3ddev, 435.5, _Form_vtxHeightFromCoord(435.5, 177.5)+2, 177.5);
FormCreate(d3ddev, 435.0, _Form_vtxHeightFromCoord(435.0, 177.5)+2, 177.5);
FormCreate(d3ddev, 434.5, _Form_vtxHeightFromCoord(434.5, 177.5)+2, 177.5);
FormCreate(d3ddev, 434.0, _Form_vtxHeightFromCoord(434.0, 177.5)+2, 177.5);
FormCreate(d3ddev, 433.5, _Form_vtxHeightFromCoord(433.5, 177.5)+2, 177.5);
FormCreate(d3ddev, 433.0, _Form_vtxHeightFromCoord(433.0, 177.5)+2, 177.5);
FormCreate(d3ddev, 432.5, _Form_vtxHeightFromCoord(432.5, 177.5)+2, 177.5);
FormCreate(d3ddev, 432.0, _Form_vtxHeightFromCoord(432.0, 177.5)+2, 177.5);
FormCreate(d3ddev, 431.5, _Form_vtxHeightFromCoord(431.5, 177.5)+2, 177.5);
FormCreate(d3ddev, 431.0, _Form_vtxHeightFromCoord(431.0, 177.5)+2, 177.5);

FormCreate(d3ddev, 436.5, _Form_vtxHeightFromCoord(436.5, 177.0)+2, 177.0);
FormCreate(d3ddev, 436.0, _Form_vtxHeightFromCoord(436.0, 177.0)+2, 177.0);
FormCreate(d3ddev, 435.5, _Form_vtxHeightFromCoord(435.5, 177.0)+2, 177.0);
FormCreate(d3ddev, 435.0, _Form_vtxHeightFromCoord(435.0, 177.0)+2, 177.0);
FormCreate(d3ddev, 434.5, _Form_vtxHeightFromCoord(434.5, 177.0)+2, 177.0);
FormCreate(d3ddev, 434.0, _Form_vtxHeightFromCoord(434.0, 177.0)+2, 177.0);
FormCreate(d3ddev, 433.5, _Form_vtxHeightFromCoord(433.5, 177.0)+2, 177.0);
FormCreate(d3ddev, 433.0, _Form_vtxHeightFromCoord(433.0, 177.0)+2, 177.0);
FormCreate(d3ddev, 432.5, _Form_vtxHeightFromCoord(432.5, 177.0)+2, 177.0);
FormCreate(d3ddev, 432.0, _Form_vtxHeightFromCoord(432.0, 177.0)+2, 177.0);
FormCreate(d3ddev, 431.5, _Form_vtxHeightFromCoord(431.5, 177.0)+2, 177.0);
FormCreate(d3ddev, 431.0, _Form_vtxHeightFromCoord(431.0, 177.0)+2, 177.0);

FormCreate(d3ddev, 436.5, _Form_vtxHeightFromCoord(436.5, 176.5)+2, 176.5);
FormCreate(d3ddev, 436.0, _Form_vtxHeightFromCoord(436.0, 176.5)+2, 176.5);
FormCreate(d3ddev, 435.5, _Form_vtxHeightFromCoord(435.5, 176.5)+2, 176.5);
FormCreate(d3ddev, 435.0, _Form_vtxHeightFromCoord(435.0, 176.5)+2, 176.5);
FormCreate(d3ddev, 434.5, _Form_vtxHeightFromCoord(434.5, 176.5)+2, 176.5);
FormCreate(d3ddev, 434.0, _Form_vtxHeightFromCoord(434.0, 176.5)+2, 176.5);
FormCreate(d3ddev, 433.5, _Form_vtxHeightFromCoord(433.5, 176.5)+2, 176.5);
FormCreate(d3ddev, 433.0, _Form_vtxHeightFromCoord(433.0, 176.5)+2, 176.5);
FormCreate(d3ddev, 432.5, _Form_vtxHeightFromCoord(432.5, 176.5)+2, 176.5);
FormCreate(d3ddev, 432.0, _Form_vtxHeightFromCoord(432.0, 176.5)+2, 176.5);
FormCreate(d3ddev, 431.5, _Form_vtxHeightFromCoord(431.5, 176.5)+2, 176.5);
FormCreate(d3ddev, 431.0, _Form_vtxHeightFromCoord(431.0, 176.5)+2, 176.5);
FormCreate(d3ddev, 430.5, _Form_vtxHeightFromCoord(430.5, 176.5)+2, 176.5);

FormCreate(d3ddev, 436.5, _Form_vtxHeightFromCoord(436.5, 176.0)+2, 176.0);
FormCreate(d3ddev, 436.0, _Form_vtxHeightFromCoord(436.0, 176.0)+2, 176.0);
FormCreate(d3ddev, 435.5, _Form_vtxHeightFromCoord(435.5, 176.0)+2, 176.0);
FormCreate(d3ddev, 435.0, _Form_vtxHeightFromCoord(435.0, 176.0)+2, 176.0);
FormCreate(d3ddev, 434.5, _Form_vtxHeightFromCoord(434.5, 176.0)+2, 176.0);
FormCreate(d3ddev, 434.0, _Form_vtxHeightFromCoord(434.0, 176.0)+2, 176.0);
FormCreate(d3ddev, 433.5, _Form_vtxHeightFromCoord(433.5, 176.0)+2, 176.0);
FormCreate(d3ddev, 433.0, _Form_vtxHeightFromCoord(433.0, 176.0)+2, 176.0);
FormCreate(d3ddev, 432.5, _Form_vtxHeightFromCoord(432.5, 176.0)+2, 176.0);
FormCreate(d3ddev, 432.0, _Form_vtxHeightFromCoord(432.0, 176.0)+2, 176.0);
FormCreate(d3ddev, 431.5, _Form_vtxHeightFromCoord(431.5, 176.0)+2, 176.0);
FormCreate(d3ddev, 431.0, _Form_vtxHeightFromCoord(431.0, 176.0)+2, 176.0);
FormCreate(d3ddev, 430.5, _Form_vtxHeightFromCoord(430.5, 176.0)+2, 176.0);

FormCreate(d3ddev, 436.5, _Form_vtxHeightFromCoord(436.5, 175.5)+2, 175.5);
FormCreate(d3ddev, 436.0, _Form_vtxHeightFromCoord(436.0, 175.5)+2, 175.5);
FormCreate(d3ddev, 435.5, _Form_vtxHeightFromCoord(435.5, 175.5)+2, 175.5);
FormCreate(d3ddev, 435.0, _Form_vtxHeightFromCoord(435.0, 175.5)+2, 175.5);
FormCreate(d3ddev, 434.5, _Form_vtxHeightFromCoord(434.5, 175.5)+2, 175.5);
FormCreate(d3ddev, 434.0, _Form_vtxHeightFromCoord(434.0, 175.5)+2, 175.5);
FormCreate(d3ddev, 433.5, _Form_vtxHeightFromCoord(433.5, 175.5)+2, 175.5);
FormCreate(d3ddev, 433.0, _Form_vtxHeightFromCoord(433.0, 175.5)+2, 175.5);
FormCreate(d3ddev, 432.5, _Form_vtxHeightFromCoord(432.5, 175.5)+2, 175.5);
FormCreate(d3ddev, 432.0, _Form_vtxHeightFromCoord(432.0, 175.5)+2, 175.5);
FormCreate(d3ddev, 431.5, _Form_vtxHeightFromCoord(431.5, 175.5)+2, 175.5);
FormCreate(d3ddev, 431.0, _Form_vtxHeightFromCoord(431.0, 175.5)+2, 175.5);
```

```
        FormCreate(d3ddev, 430.5, _Form_vtxHeightFromCoord(430.5, 175.5)+2, 175.5);
        FormCreate(d3ddev, 430.0, _Form_vtxHeightFromCoord(430.0, 175.5)+2, 175.5);

        FormCreate(d3ddev, 436.5, _Form_vtxHeightFromCoord(436.5, 175.0)+2, 175.0);
        FormCreate(d3ddev, 436.0, _Form_vtxHeightFromCoord(436.0, 175.0)+2, 175.0);
        FormCreate(d3ddev, 435.5, _Form_vtxHeightFromCoord(435.5, 175.0)+2, 175.0);
        FormCreate(d3ddev, 435.0, _Form_vtxHeightFromCoord(435.0, 175.0)+2, 175.0);
        FormCreate(d3ddev, 434.5, _Form_vtxHeightFromCoord(434.5, 175.0)+2, 175.0);
        FormCreate(d3ddev, 434.0, _Form_vtxHeightFromCoord(434.0, 175.0)+2, 175.0);
        FormCreate(d3ddev, 433.5, _Form_vtxHeightFromCoord(433.5, 175.0)+2, 175.0);
        FormCreate(d3ddev, 433.0, _Form_vtxHeightFromCoord(433.0, 175.0)+2, 175.0);
        FormCreate(d3ddev, 432.5, _Form_vtxHeightFromCoord(432.5, 175.0)+2, 175.0);
        FormCreate(d3ddev, 432.0, _Form_vtxHeightFromCoord(432.0, 175.0)+2, 175.0);
        FormCreate(d3ddev, 431.5, _Form_vtxHeightFromCoord(431.5, 175.0)+2, 175.0);
        FormCreate(d3ddev, 431.0, _Form_vtxHeightFromCoord(431.0, 175.0)+2, 175.0);
        FormCreate(d3ddev, 430.5, _Form_vtxHeightFromCoord(430.5, 175.0)+2, 175.0);
        FormCreate(d3ddev, 430.0, _Form_vtxHeightFromCoord(430.0, 175.0)+2, 175.0);

        FormCreate(d3ddev, 436.5, _Form_vtxHeightFromCoord(436.5, 174.5)+2, 174.5);
        FormCreate(d3ddev, 436.0, _Form_vtxHeightFromCoord(436.0, 174.5)+2, 174.5);
        FormCreate(d3ddev, 435.5, _Form_vtxHeightFromCoord(435.5, 174.5)+2, 174.5);
        FormCreate(d3ddev, 435.0, _Form_vtxHeightFromCoord(435.0, 174.5)+2, 174.5);
        FormCreate(d3ddev, 434.5, _Form_vtxHeightFromCoord(434.5, 174.5)+2, 174.5);
        FormCreate(d3ddev, 434.0, _Form_vtxHeightFromCoord(434.0, 174.5)+2, 174.5);
        FormCreate(d3ddev, 433.5, _Form_vtxHeightFromCoord(433.5, 174.5)+2, 174.5);
        FormCreate(d3ddev, 433.0, _Form_vtxHeightFromCoord(433.0, 174.5)+2, 174.5);
        FormCreate(d3ddev, 432.5, _Form_vtxHeightFromCoord(432.5, 174.5)+2, 174.5);
        FormCreate(d3ddev, 432.0, _Form_vtxHeightFromCoord(432.0, 174.5)+2, 174.5);
        FormCreate(d3ddev, 431.5, _Form_vtxHeightFromCoord(431.5, 174.5)+2, 174.5);
        FormCreate(d3ddev, 431.0, _Form_vtxHeightFromCoord(431.0, 174.5)+2, 174.5);
        FormCreate(d3ddev, 430.5, _Form_vtxHeightFromCoord(430.5, 174.5)+2, 174.5);
        FormCreate(d3ddev, 430.0, _Form_vtxHeightFromCoord(430.0, 174.5)+2, 174.5);

        FormCreate(d3ddev, 436.5, _Form_vtxHeightFromCoord(436.5, 174.0)+2, 174.0);
        FormCreate(d3ddev, 436.0, _Form_vtxHeightFromCoord(436.0, 174.0)+2, 174.0);
        FormCreate(d3ddev, 435.5, _Form_vtxHeightFromCoord(435.5, 174.0)+2, 174.0);
        FormCreate(d3ddev, 435.0, _Form_vtxHeightFromCoord(435.0, 174.0)+2, 174.0);
        FormCreate(d3ddev, 434.5, _Form_vtxHeightFromCoord(434.5, 174.0)+2, 174.0);
        FormCreate(d3ddev, 434.0, _Form_vtxHeightFromCoord(434.0, 174.0)+2, 174.0);
        FormCreate(d3ddev, 433.5, _Form_vtxHeightFromCoord(433.5, 174.0)+2, 174.0);
        FormCreate(d3ddev, 433.0, _Form_vtxHeightFromCoord(433.0, 174.0)+2, 174.0);
        FormCreate(d3ddev, 432.5, _Form_vtxHeightFromCoord(432.5, 174.0)+2, 174.0);
        FormCreate(d3ddev, 432.0, _Form_vtxHeightFromCoord(432.0, 174.0)+2, 174.0);
        FormCreate(d3ddev, 431.5, _Form_vtxHeightFromCoord(431.5, 174.0)+2, 174.0);
        FormCreate(d3ddev, 431.0, _Form_vtxHeightFromCoord(431.0, 174.0)+2, 174.0);
        FormCreate(d3ddev, 430.5, _Form_vtxHeightFromCoord(430.5, 174.0)+2, 174.0);
        FormCreate(d3ddev, 430.0, _Form_vtxHeightFromCoord(430.0, 174.0)+2, 174.0);

        FormCreate(d3ddev, 0.0, _Form_vtxHeightFromCoord(0.0, 0.0)+5, 0.0);

}

#endif
```

5.9 hmap.h

```
#ifndef HMAP
#define HMAP

//includes
#include<stdio.h>
#include<stdlib.h>
#include<fstream>
#include<d3d9.h>
#include<d3dx9.h>
#include<string>
#include"global_defines.h"
#include"hmap_defines.h"
#include"color_defines.h"
#include<cstdio>
#include"string_udf.h"
```

```
//structs & classes
class BMP
{
public:
        BITMAPFILEHEADER FileHeader;
        BITMAPINFOHEADER InfoHeader;
        unsignedchar *BMPData;
        int error;
};

class HMap_class {
public:
        BMP file;                       //file class
        LPDIRECT3DTEXTURE9 texture;     //texture
        bool txt_overall;               //texture over whole hmap or not
        D3DMATERIAL9 material;          //material
        VERTEX_3D* vertices;            //array holding vertex information
        unsignedint* indices;           //array holding index information
        int width;                      //h-map width
        int height;                     //h-map height
        int size;                       //h-map size
        LPDIRECT3DVERTEXBUFFER9 v_buffer;       //vertex buffer
        LPDIRECT3DINDEXBUFFER9 i_buffer;        //index buffer
        bool state;                             //rendering state
        //std::string filename; //h-map file name
        //std::string texturename;//texture file name
        int id;         //table id (DHM25 / DHM200)

        int type;       //0=DHM25 / 1=DHM200 / 2=File
};

HMap_class HMap;//the class to the Height-Map

//function prototypes
void initHMap(LPDIRECT3DDEVICE9 d3ddev);
//bool LoadHMap (LPDIRECT3DDEVICE9 d3ddev, const char *Filename, const char *Texturename);
bool HMap_DHM200 (LPDIRECT3DDEVICE9 d3ddev, int id);
bool HMap_DHM25 (LPDIRECT3DDEVICE9 d3ddev, int id);
bool HMap_texture (LPDIRECT3DDEVICE9 d3ddev, constchar* file);
void HMap_reload (LPDIRECT3DDEVICE9 d3ddev);

BMP LoadBitmapFile(constchar *Filename);

int _HMap_imgListfromCoord (int x, int y); //gets the quad number of the h-map at a certain position
int _HMap_vtxListfromCoord (int x, int y); //gets the quad number of a vertex at a certain position
void _HMap_VertexPos(VERTEX_3D &vertex, float x, float y, float height); //sets x,y,z of a vertex
void _HMap_VertexTexPos(VERTEX_3D &vertex, float x, float y);
void _HMap_VertexNormal(int x, int y);

/*bool LoadHMap (LPDIRECT3DDEVICE9 d3ddev, const char *Filename, const char *Texturename) {

        HMap.filename = Filename; //name of height-map file
        HMap.texturename = Texturename; //name of texture file

        char buffer1[255];
        char buffer2[255];
        sprintf(buffer1,"Maps\\hmap\\%s", Filename);
        Filename = buffer1; //"const char*" to "char" + path
        sprintf(buffer2,"Maps\\texture\\%s", Texturename);
        Texturename = buffer2;//"const char*" to "char" + path



        HMap.file = LoadBitmapFile(Filename); //load height-map file data
        if(HMap.file.error != 1) { return 0;};

        HMap.texture = LoadBitmapFile(Texturename); //load textuer file data
        if(HMap.file.error != 1) { return 0;};

        HMap.width = HMap.file.InfoHeader.biHeight; //hmap width
        HMap.size = (HMap.width*HMap.width) - (2*HMap.width); //hmap size
        HMAP_WIDTH = HMap.width;
        HMAP_SIZE = HMap.size;

        HMap.vertices = new VERTEX_3D[HMap.size]; //vertex array (holding all coordinate + height)
        HMap.indices = new unsigned int[HMap.size*2]; //index array (holding vertex drawing order)
```

```
	//write BMPData into vertices array
	int i=0;
	for (int y=1 ; y <= HMAP_WIDTH-1 ; y++){
	for(int x=1 ; x <= HMAP_WIDTH-1 ; x++){
		//set coordinates + height (from hmap file)
		VertexPos(HMap.vertices[i], (float) x, (float) y, (HMap.file.BMPData[imgListfromCoord(x,y)]/HMAP_DIV));
		VertexColor(HMap.vertices[i], imgColorfromCoord(x, y)); //set color (from textuer file)
		i+=1;
	}}

	//write vertex data into index array
	i=0;
	for (int y=1 ; y <= HMAP_WIDTH-1 ; y++){
		if (y%2==1) { //uneven row -> from bottom to top
			for(int x=1 ; x <= HMAP_WIDTH-1 ; x++){
				HMap.indices[i] = vtxListfromCoord(x, y+1); //write index number in index array
				if(x != HMAP_WIDTH-1) {HMap.indices[i+1] = vtxListfromCoord(x+1, y);}
				else {HMap.indices[i+1] = vtxListfromCoord(x, y);}
				i+=2;
			}
		} else { //even row -> from top to bottom
			for(int x=HMAP_WIDTH-1; x >= 1 ; x--){
				HMap.indices[i] = vtxListfromCoord(x, y+1); //write index number in index array
				if(x != 1) {HMap.indices[i+1] = vtxListfromCoord(x-1, y);}
				else {HMap.indices[i+1] = vtxListfromCoord(x, y);}
				i+=2;
			}
		}
	}

	//vertex buffer holding vertex array
	d3ddev->CreateVertexBuffer(HMap.size*sizeof(VERTEX_3D), 0, FVF_3D, D3DPOOL_MANAGED, &HMap.v_buffer, NULL);

	VOID* pVoid;    // a void pointer
	HMap.v_buffer->Lock(0, 0, (void**)&pVoid, 0); // lock vertex buffer
	memcpy(pVoid, HMap.vertices, HMap.size*sizeof(VERTEX_3D)); //load vertices into it
	HMap.v_buffer->Unlock(); //unlock

	//index buffer holding index array
	d3ddev->CreateIndexBuffer(HMap.size*2*sizeof(unsigned int), 0, D3DFMT_INDEX32, D3DPOOL_MANAGED,
	&HMap.i_buffer,NULL);

	HMap.i_buffer->Lock(0, 0, (void**)&pVoid, 0);// lock index buffer
	memcpy(pVoid, HMap.indices, HMap.size*2*sizeof(unsigned int)); //load indices into it
	HMap.i_buffer->Unlock(); //unlock


	HMap.state = 1; //OK for drawing

	return 1;

}*/

bool HMap_DHM200 (LPDIRECT3DDEVICE9 d3ddev, int id) {

	HMap.id = id;
	HMap.type = 1;

	char filename[255];
	char line [40];
	FILE * file;

	sprintf_s(filename, "Terrain\\DHM\\200\\%d.csv", id);
	if (fopen_s(&file, filename,"r") != 0){ return 0;}

	sprintf_s(filename, "Terrain\\DHM\\table.csv");
	fopen_s(&file, filename,"r");

	std::string a_param[5]; //string array to hold table paramter

	while( fgets(line, sizeof line, file) != NULL) {
		std::list<std::string> l_param; //list to hold table parameter
		split(line,';', l_param); //split table line into parameter
		int i = 0; //counter
		for (std::list<std::string>::iterator it = l_param.begin(); it != l_param.end(); it++) { //for all
```

```
                a_param[i] = (std::string) *it; //translate list into array (easier handling)
                i++;
        }
        if (atof(a_param[0].c_str()) == id) {
                break;
        }

}
fclose(file);

HMap.width = (int) (atof(a_param[2].c_str())-atof(a_param[1].c_str()))/200; //hmap width
HMap.height = (int) (atof(a_param[4].c_str())-atof(a_param[3].c_str()))/200; //hmap height
HMap.size = HMap.width*HMap.height; //hmap size
HMAP_WIDTH = HMap.width;
HMAP_HEIGHT = HMap.height;
HMAP_SIZE = HMap.size;

HMap.vertices = new VERTEX_3D[HMap.size]; //vertex array (holding all coordinate + height)
HMap.indices = newunsignedint[HMap.size*2]; //index array (holding vertex drawing order)

sprintf_s(filename, "Terrain\\DHM\\200\\%d.csv", id);
fopen_s(&file, filename,"r");
//write data into vertices array
int i=0;
for (int y=1 ; y <= HMAP_HEIGHT ; y++){
for(int x=1 ; x <= HMAP_WIDTH ; x++){
        fgets(line, sizeof line, file);
        //set coordinates + height (from hmap file)
        _HMap_VertexPos(HMap.vertices[i], (float) (x*-1)+HMAP_WIDTH, (float) y, (float) atof(line)/HMAP_DIV);
        _HMap_VertexTexPos(HMap.vertices[i], (float) x, (float) y);
        i+=1;
}}
fclose(file);

for (int y=1 ; y <= HMAP_HEIGHT ; y++){
for(int x=1 ; x <= HMAP_WIDTH ; x++){
        _HMap_VertexNormal(x, y);
}}

//write vertex data into index array
i=0;
for (int y=1 ; y <= HMAP_HEIGHT ; y++){
        if (y%2==1) { //uneven row -> from bottom to top
                for(int x=1 ; x <= HMAP_WIDTH ; x++){
                        HMap.indices[i] = _HMap_vtxListfromCoord(x, y+1); //write index number in index array
                        if(x != HMAP_WIDTH) {HMap.indices[i+1] = _HMap_vtxListfromCoord(x+1, y);}
                        else {HMap.indices[i+1] = _HMap_vtxListfromCoord(x, y);}
                        i+=2;
                }
        } else { //even row -> from top to bottom
                for(int x=HMAP_WIDTH; x >= 1 ; x--){
                        HMap.indices[i] = _HMap_vtxListfromCoord(x, y+1); //write index number in index array
                        if(x != 1) {HMap.indices[i+1] = _HMap_vtxListfromCoord(x-1, y);}
                        else {HMap.indices[i+1] = _HMap_vtxListfromCoord(x, y);}
                        i+=2;
                }
        }
}

//vertex buffer holding vertex array
d3ddev->CreateVertexBuffer(HMap.size*sizeof(VERTEX_3D), 0, FVF_3D, D3DPOOL_MANAGED, &HMap.v_buffer, NULL);

VOID* pVoid;    // a void pointer
HMap.v_buffer->Lock(0, 0, (void**)&pVoid, 0); // lock vertex buffer
memcpy(pVoid, HMap.vertices, HMap.size*sizeof(VERTEX_3D)); //load vertices into it
HMap.v_buffer->Unlock(); //unlock

//index buffer holding index array
d3ddev->CreateIndexBuffer(HMap.size*2*sizeof(unsignedint), 0, D3DFMT_INDEX32, D3DPOOL_MANAGED,
&HMap.i_buffer,NULL);

HMap.i_buffer->Lock(0, 0, (void**)&pVoid, 0);// lock index buffer
memcpy(pVoid, HMap.indices, HMap.size*2*sizeof(unsignedint)); //load indices into it
HMap.i_buffer->Unlock(); //unlock
```

```
	HMap.state = 1; //OK for drawing

	return 1;

}

bool HMap_DHM25 (LPDIRECT3DDEVICE9 d3ddev, int id) {

	HMap.id = id;
	HMap.type = 0;

	char filename[255];
	char line [40];
	FILE * file;

	sprintf_s(filename, "Terrain\\DHM\\25\\%d.csv", id);
	if (fopen_s(&file, filename,"r") != 0){ return 0;}

	sprintf_s(filename, "Terrain\\DHM\\table.csv");
	fopen_s(&file, filename,"r");

	std::string a_param[5]; //string array to hold table paramter

	while( fgets(line, sizeof line, file) != NULL) {
		std::list<std::string> l_param; //list to hold table parameter
		split(line,';', l_param); //split table line into parameter
		int i = 0; //counter
		for (std::list<std::string>::iterator it = l_param.begin(); it != l_param.end(); it++) { //for all
			a_param[i] = (std::string) *it; //translate list into array (easier handling)
			i++;
		}
		if (atof(a_param[0].c_str()) == id) {
			break;
		}

	}
	fclose(file);

	HMap.width = (int) (atof(a_param[2].c_str())-atof(a_param[1].c_str()))/25+1; //hmap width
	HMap.height = (int) (atof(a_param[4].c_str())-atof(a_param[3].c_str()))/25+1; //hmap height
	HMap.size = HMap.width*HMap.height; //hmap size
	HMAP_WIDTH = HMap.width;
	HMAP_HEIGHT = HMap.height;
	HMAP_SIZE = HMap.size;

	HMap.vertices = new VERTEX_3D[HMap.size]; //vertex array (holding all coordinate + height)
	HMap.indices = newunsignedint[HMap.size*2]; //index array (holding vertex drawing order)


	sprintf_s(filename, "Terrain\\DHM\\25\\%d.csv", id);
	fopen_s(&file, filename,"r");
	//write data into vertices array
	int i=0;
	for (int y=1 ; y <= HMAP_HEIGHT ; y++){
	for(int x=1 ; x <= HMAP_WIDTH ; x++){
		fgets(line, sizeof line, file);
		//set coordinates + height (from hmap file)
		_HMap_VertexPos(HMap.vertices[i], (float) (x*-1)+HMAP_WIDTH, (float) y, (float) (atof(line)/HMAP_DIV));
		_HMap_VertexTexPos(HMap.vertices[i], (float) x, (float) y);
		i+=1;
	}}

	fclose(file);

	for (int y=1 ; y <= HMAP_HEIGHT ; y++){
	for(int x=1 ; x <= HMAP_WIDTH ; x++){
		_HMap_VertexNormal(x, y);
	}}

	//write vertex data into index array
	i=0;
	for (int y=1 ; y <= HMAP_HEIGHT ; y++){
		if (y%2==1) { //uneven row -> from bottom to top
			for(int x=1 ; x <= HMAP_WIDTH ; x++){
				HMap.indices[i] = _HMap_vtxListfromCoord(x, y+1); //write index number in index array
				if(x != HMAP_WIDTH) {HMap.indices[i+1] = _HMap_vtxListfromCoord(x+1, y);}
```

```
                                else {HMap.indices[i+1] = _HMap_vtxListfromCoord(x, y);}
                                i+=2;
                        }
                } else { //even row -> from top to bottom
                        for(int x=HMAP_WIDTH; x >= 1 ; x--){
                                HMap.indices[i] = _HMap_vtxListfromCoord(x, y+1); //write index number in index array
                                if(x != 1) {HMap.indices[i+1] = _HMap_vtxListfromCoord(x-1, y);}
                                else {HMap.indices[i+1] = _HMap_vtxListfromCoord(x, y);}
                                i+=2;
                        }
                }
        }

        //vertex buffer holding vertex array
        d3ddev->CreateVertexBuffer(HMap.size*sizeof(VERTEX_3D), 0, FVF_3D, D3DPOOL_MANAGED, &HMap.v_buffer, NULL);

        VOID* pVoid;    // a void pointer
        HMap.v_buffer->Lock(0, 0, (void**)&pVoid, 0); // lock vertex buffer
        memcpy(pVoid, HMap.vertices, HMap.size*sizeof(VERTEX_3D)); //load vertices into it
        HMap.v_buffer->Unlock(); //unlock

        //index buffer holding index array
        d3ddev->CreateIndexBuffer(HMap.size*2*sizeof(unsignedint), 0, D3DFMT_INDEX32, D3DPOOL_MANAGED,
&HMap.i_buffer,NULL);

        HMap.i_buffer->Lock(0, 0, (void**)&pVoid, 0);// lock index buffer
        memcpy(pVoid, HMap.indices, HMap.size*2*sizeof(unsignedint)); //load indices into it
        HMap.i_buffer->Unlock(); //unlock

        HMap.state = 1; //OK for drawing

        return 1;

}

bool HMap_texture (LPDIRECT3DDEVICE9 d3ddev,  constchar* file){
        char buffer[255];

        if (file == "spot") {
                if (HMap.txt_overall == false) {
                        HMap.txt_overall = true;
                        HMap_reload(d3ddev);
                }
                sprintf_s(buffer,"Terrain\\Texture\\spot\\%d.png", HMap.id);
        } else {
                if (HMap.txt_overall == true) {
                        HMap.txt_overall = false;
                        HMap_reload(d3ddev);
                }
                sprintf_s(buffer,"Terrain\\Texture\\%s.png",file);
        }

        if (D3DXCreateTextureFromFile(d3ddev, buffer, &HMap.texture) != D3D_OK){ return 0;}

        return 1;
}

void HMap_reload (LPDIRECT3DDEVICE9 d3ddev) {
        switch (HMap.type) {
                case 0: //DHM25
                        HMap_DHM25(d3ddev, HMap.id);
                        break;
                case 1: //DHM200
                        HMap_DHM200(d3ddev, HMap.id);
                        break;
                case 2: //File

                        break;
        }

}

void initHMap(LPDIRECT3DDEVICE9 d3ddev) {
        HMap.state = 0; //dont draw
        HMap.material.Diffuse = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f);    // set diffuse color to white
        HMap.material.Ambient = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f);    // set ambient color to white
```

```
        HMap.txt_overall = false;
}

BMP LoadBitmapFile(constchar *Filename)
{

        /*----!!! IMPORTANT !!!-----
        BMP File must match this requirements:
         - Not Comprimed (24bpp)
         - Height and Width must be multiples of 4.
        */

        BMP Bitmap;

        FILE *File;
        int imageIdx=0;

        char buffer[255];
        sprintf_s(buffer, "Terrain\\DHM\\25\\%d.csv", Filename);
        fopen_s(&File, buffer,"rb");

        if (File == NULL)
        {
                Bitmap.error = -1;
                return Bitmap;
        }

        // Read the file header
        fread(&(Bitmap.FileHeader), sizeof(BITMAPFILEHEADER), 1, File);

        // Check if its a bitmap or not
        if (Bitmap.FileHeader.bfType != 0x4D42)
        {
                Bitmap.error = -2;
                return Bitmap;
        }

        // Read the info header
        fread(&Bitmap.InfoHeader, sizeof(BITMAPINFOHEADER), 1, File);

        Bitmap.InfoHeader.biSizeImage = Bitmap.InfoHeader.biHeight * Bitmap.InfoHeader.biWidth * Bit-
        map.InfoHeader.biBitCount/8;


        //move file point to the begging of bitmap data
        fseek(File, Bitmap.FileHeader.bfOffBits, SEEK_SET);

        // Allocate memory
        Bitmap.BMPData = (unsignedchar*)malloc(Bitmap.InfoHeader.biSizeImage);

        if (!Bitmap.BMPData)
        {
                fclose(File);
                Bitmap.error = -3;
                return Bitmap;
        }

        //read in the bitmap image data
        fread(Bitmap.BMPData, Bitmap.InfoHeader.biSizeImage, 1, File);

        //make sure bitmap image data was read
        if (Bitmap.BMPData == NULL)
        {
                fclose(File);
                Bitmap.error = -4;
                return Bitmap;
        }

        /*for (imageIdx = 0 ; imageIdx < Bitmap.InfoHeader.biSizeImage ; imageIdx+=3)
        {
                Bitmap.BMPData[imageIdx/3] = Bitmap.BMPData[imageIdx];
        }*/


        Bitmap.InfoHeader.biSizeImage = Bitmap.InfoHeader.biHeight * Bitmap.InfoHeader.biWidth;
```

```
        // Clean up and return
        fclose(File);
        Bitmap.error = 1;
        return Bitmap;
};


int _HMap_imgListfromCoord (int x, int y) { //get list id for BMP.BMPData[id] from x,y coords
        return ((y-1)*HMAP_WIDTH+x-1)*3;
}


int _HMap_vtxListfromCoord (int x, int y) { //get list id for HMap.vertices[id] from x,y coord
        return ((y-1)*(HMAP_WIDTH)+x-1);
}

void _HMap_VertexPos(VERTEX_3D &vertex, float x, float y, float z) {     //set vertex x,y,z coords
        vertex.X = x;
        vertex.Y = z;
        vertex.Z = y;
}

void _HMap_VertexTexPos(VERTEX_3D &vertex, float x, float y) { //set vertex texture position
        if (HMap.txt_overall == true) { //if overall texture (i.e. spot)
                vertex.U = x/HMAP_WIDTH;
                vertex.V = y/HMAP_HEIGHT;
        } else {        //else: one texture for every square
                vertex.U = x;
                vertex.V = y;
        }
}

void _HMap_VertexNormal(int x, int y) { //set vertex normal (light calc)

        int vector_0_list = _HMap_vtxListfromCoord(x,y),
                vector_1_list = _HMap_vtxListfromCoord(x+1,y),
                vector_2_list = _HMap_vtxListfromCoord(x-1,y),
                vector_3_list = _HMap_vtxListfromCoord(x,y+1),
                vector_4_list = _HMap_vtxListfromCoord(x,y-1);

        D3DXVECTOR3 normal0 (0, 0, 0), normal1 (0, 0, 0), normal2 (0, 0, 0), normal3 (0, 0, 0), normal4 (0, 0, 0);
        D3DXVECTOR3 vector1 (0, 0, 0), vector2 (0, 0, 0), vector3 (0, 0, 0), vector4 (0, 0, 0);
        D3DXVECTOR3 vector0 (HMap.vertices[vector_0_list].X, HMap.vertices[vector_0_list].Z,
        HMap.vertices[vector_0_list].Y);

        if(x+1<=HMAP_WIDTH) { //vector 1
                vector1.x = HMap.vertices[vector_1_list].X;
                vector1.y = HMap.vertices[vector_1_list].Z;
                vector1.z = HMap.vertices[vector_1_list].Y;
                vector1 = vector1-vector0;
        }
        if(x-1>=1) { //vector 2
                vector2.x = HMap.vertices[vector_2_list].X;
                vector2.y = HMap.vertices[vector_2_list].Z;
                vector2.z = HMap.vertices[vector_2_list].Y;
                vector2 = vector2-vector0;
        }
        if(y+1<=HMAP_HEIGHT) { //vector 3
                vector3.x = HMap.vertices[vector_3_list].X;
                vector3.y = HMap.vertices[vector_3_list].Z;
                vector3.z = HMap.vertices[vector_3_list].Y;
                vector3 = vector3-vector0;
        }
        if(y-1>=1) { //vector 4
                vector4.x = HMap.vertices[vector_4_list].X;
                vector4.y = HMap.vertices[vector_4_list].Z;
                vector4.z = HMap.vertices[vector_4_list].Y;
                vector4 = vector4-vector0;
        }

        if (!(vector4.x==0 && vector4.y==0 && vector4.z==0) && !(vector1.x==0 && vector1.y==0 && vector1.z==0)) {
                D3DXVec3Cross(&normal1, &vector4, &vector1);
                D3DXVec3Normalize(&normal1, &normal1);
        }
        if (!(vector1.x==0 && vector1.y==0 && vector1.z==0) && !(vector3.x==0 && vector3.y==0 && vector3.z==0)) {
```

```
                D3DXVec3Cross(&normal2, &vector1, &vector3);
                D3DXVec3Normalize(&normal2, &normal2);
        }
        if (!(vector3.x==0 && vector3.y==0 && vector3.z==0) && !(vector2.x==0 && vector2.y==0 && vector2.z==0)) {
                D3DXVec3Cross(&normal3, &vector3, &vector2);
                D3DXVec3Normalize(&normal3, &normal3);
        }
        if (!(vector2.x==0 && vector2.y==0 && vector2.z==0) && !(vector4.x==0 && vector4.y==0 && vector4.z==0)) {
                D3DXVec3Cross(&normal4, &vector2, &vector4);
                D3DXVec3Normalize(&normal4, &normal4);
        }

        int i=0;
        if(!(normal1.x==0 && normal1.y==0 && normal1.z==0)){
                normal0 += normal1;
                i+=1;
        }
        if(!(normal2.x==0 && normal2.y==0 && normal2.z==0)){
                normal0 += normal2;
                i+=1;
        }
        if(!(normal3.x==0 && normal3.y==0 && normal3.z==0)){
                normal0 += normal3;
                i+=1;
        }
        if(!(normal4.x==0 && normal4.y==0 && normal4.z==0)){
                normal0 += normal4;
                i+=1;
        }

        normal0 = normal0/i;

        HMap.vertices[vector_0_list].NORMAL.x = -normal0.x;
        HMap.vertices[vector_0_list].NORMAL.y = -normal0.z;
        HMap.vertices[vector_0_list].NORMAL.z = -normal0.y;
}

#endif
```

5.10 string_udf.h

```
#ifndef string_udf
#define sting_udf

#include<string>
#include<list>


std::string UpToLow(std::string str);
std::string LowToUP(std::string str);
staticvoid split(const std::string& str, char d, std::list<std::string>& list);
std::string rem_space(std::string str);


std::string UpToLow(std::string str) {
        for (unsignedint i=0;i<strlen(str.c_str());i++) {
                if (str[i] >= 0x41 && str[i] <= 0x5A)
                        str[i] = str[i] + 0x20;
        }

        return str;
}

std::string LowToUp(std::string str) {
        for (unsignedint i=0;i<strlen(str.c_str());i++) {
                if (str[i] >= 0x61 && str[i] <= 0x7A)
                        str[i] = str[i] - 0x20;
        }

        return str;
}
```

```
staticvoid split(const std::string& str, char d, std::list<std::string>& list){

	constchar* s(str.c_str());
	while(*s){
		constchar* item(s);
		while(*s && *s != d) s++; // only mv s fwd if we are not at string end
		list.push_back(std::string(item, s-item)); // template over list if needed
		if (*s && !*(++s)) list.push_back("");
	}
}

std::string rem_space(std::string str) {

	for (unsignedint i = 0; i < str.length(); i++) {
		if (str[i] == ' ')
			str.erase(i, 1);
	}
	return str;
}

#endif
```

5.11 env.h

```
#ifndef env
#define env

class env_class {
public:
	LPD3DXMESH skysphere;
	LPDIRECT3DTEXTURE9 skysphere_txt;
};

env_class Env;

bool SkySphere_txt(LPDIRECT3DDEVICE9 d3ddev, constchar* nr);
void SkySphere_init(LPDIRECT3DDEVICE9 d3ddev);

bool SkySphere_txt(LPDIRECT3DDEVICE9 d3ddev, constchar* nr) {
	char buffer[255];
	sprintf_s(buffer, "Environment\\SkySphere\\skysphere%s.png", nr);
	if (D3DXCreateTextureFromFileA(d3ddev, buffer, &Env.skysphere_txt) != D3D_OK) {return 0;}
	return 1;
}

void SkySphere_init(LPDIRECT3DDEVICE9 d3ddev) {
	D3DXLoadMeshFromX("Environment\\SkySphere\\skysphere.x", D3DXMESH_SYSTEMMEM, d3ddev, NULL, NULL, NULL, NULL,
	&Env.skysphere); //init skybox
}

#endif
```

5.12 gui.h

```
#ifndef GUI_h
#define GUI_h

#include<d3d9.h>
#include<d3dx9.h>
#include<windows.h>
#include"global_defines.h"

class class_gui {
public:
	int focus; //tab who has focus (0=no tabs displayed)

	VERTEX_2D vTb1[4];		//Tab 1 (Terrain)
	LPDIRECT3DVERTEXBUFFER9 bTb1;	//
```

```
        LPDIRECT3DTEXTURE9 tTb1;        //
        VERTEX_2D vTb2[4];              //Tab 2 (Texture)
        LPDIRECT3DVERTEXBUFFER9 bTb2;   //
        LPDIRECT3DTEXTURE9 tTb2;        //
        VERTEX_2D vTb3[4];              //Tab 3 (Environment)
        LPDIRECT3DVERTEXBUFFER9 bTb3;   //
        LPDIRECT3DTEXTURE9 tTb3;        //
        VERTEX_2D vTb4[4];              //Tab 4 (Forms)
        LPDIRECT3DVERTEXBUFFER9 bTb4;   //
        LPDIRECT3DTEXTURE9 tTb4;        //

};

class_gui GUI;

void GUIInit(LPDIRECT3DDEVICE9 d3ddev);
void GUIProc();

void GUIProc() {

        POINT cursorPos;
        GetCursorPos(&cursorPos);

        STARTUPINFO          si = { sizeof(si) };
        PROCESS_INFORMATION  pi;

        if      (m_MouseState.rgbButtons[0] & 0x80 &&
                cursorPos.x > SCREEN_WIDTH-50 && cursorPos.x < SCREEN_WIDTH && cursorPos.y > 10 && cursorPos.y < 60) {
                if (FindWindow("AutoIt v3 GUI", "World Terrain") == NULL){
                        CreateProcess("GUI\\terrain.exe", NULL, NULL, NULL, FALSE, NULL, NULL, NULL, &si, &pi);
                        while(FindWindow("AutoIt v3 GUI", "World Terrain") == NULL);
                }
        } elseif(m_MouseState.rgbButtons[0] & 0x80 &&
                cursorPos.x > SCREEN_WIDTH-50 && cursorPos.x < SCREEN_WIDTH && cursorPos.y > 60 && cursorPos.y < 110) {
                if (FindWindow("AutoIt v3 GUI", "World Texture") == NULL){
                        CreateProcess("GUI\\texture.exe", NULL, NULL, NULL, TRUE, CREATE_DEFAULT_ERROR_MODE, NULL, NULL,
                        &si, &pi);
                        while(FindWindow("AutoIt v3 GUI", "World texture") == NULL);
                }
        } elseif(m_MouseState.rgbButtons[0] & 0x80&&
                 cursorPos.x > SCREEN_WIDTH-50 && cursorPos.x < SCREEN_WIDTH && cursorPos.y > 110 && cursorPos.y < 160) {
                if (FindWindow("AutoIt v3 GUI", "World Environment") == NULL){
                        CreateProcess("GUI\\environment.exe", NULL, NULL, NULL, TRUE, CREATE_DEFAULT_ERROR_MODE, NULL,
                        NULL, &si, &pi);
                        while(FindWindow("AutoIt v3 GUI", "World Environment") == NULL);
                }
        } elseif(m_MouseState.rgbButtons[0] & 0x80 &&
                cursorPos.x > SCREEN_WIDTH-50 && cursorPos.x < SCREEN_WIDTH && cursorPos.y > 160 && cursorPos.y < 210) {
                if (FindWindow("AutoIt v3 GUI", "World Objects") == NULL){
                        CreateProcess("GUI\\objects.exe", NULL, NULL, NULL, TRUE, CREATE_DEFAULT_ERROR_MODE, NULL, NULL,
                        &si, &pi);
                        while(FindWindow("AutoIt v3 GUI", "World Objects") == NULL);
                }
        }
}


void GUIInit(LPDIRECT3DDEVICE9 d3ddev) {

        GUI.focus = 0;

        VOID* pVoid;

        //*** Tab 1 (Terrain) ***
        GUI.vTb1[0].X = (float) SCREEN_WIDTH-50;        GUI.vTb1[0].Y = 10.0f;          GUI.vTb1[0].Z = 0.5f;
        GUI.vTb1[0].RHW = 1.0f;         GUI.vTb1[0].U = 0;      GUI.vTb1[0].V = 0;
        GUI.vTb1[1].X = (float) SCREEN_WIDTH;           GUI.vTb1[1].Y = 10.0f;          GUI.vTb1[1].Z = 0.5f;
        GUI.vTb1[1].RHW = 1.0f;         GUI.vTb1[1].U = 1;      GUI.vTb1[1].V = 0;
        GUI.vTb1[2].X = (float) SCREEN_WIDTH-50;        GUI.vTb1[2].Y = 60.0f;          GUI.vTb1[2].Z = 0.5f;
        GUI.vTb1[2].RHW = 1.0f;         GUI.vTb1[2].U = 0;      GUI.vTb1[2].V = 1;
        GUI.vTb1[3].X = (float) SCREEN_WIDTH;           GUI.vTb1[3].Y = 60.0f;          GUI.vTb1[3].Z = 0.5f;
        GUI.vTb1[3].RHW = 1.0f;         GUI.vTb1[3].U = 1;      GUI.vTb1[3].V = 1;

        d3ddev->CreateVertexBuffer(4*sizeof(VERTEX_2D), 0, FVF_2D, D3DPOOL_MANAGED, &GUI.bTb1, NULL);
        GUI.bTb1->Lock(0, 0, (void**)&pVoid, 0);
        memcpy(pVoid, GUI.vTb1, 4*sizeof(VERTEX_2D));
```

```
        GUI.bTb1->Unlock();
        D3DXCreateTextureFromFileA(d3ddev, "GUI\\Image\\tTb1.png", &GUI.tTb1);

        //*** Tab 2 (texture) ***
        GUI.vTb2[0].X = (float) SCREEN_WIDTH-50;        GUI.vTb2[0].Y = 60.0f;          GUI.vTb2[0].Z = 0.5f;
        GUI.vTb2[0].RHW = 1.0f;         GUI.vTb2[0].U = 0;      GUI.vTb2[0].V = 0;
        GUI.vTb2[1].X = (float) SCREEN_WIDTH;           GUI.vTb2[1].Y = 60.0f;          GUI.vTb2[1].Z = 0.5f;
        GUI.vTb2[1].RHW = 1.0f;         GUI.vTb2[1].U = 1;      GUI.vTb2[1].V = 0;
        GUI.vTb2[2].X = (float) SCREEN_WIDTH-50;        GUI.vTb2[2].Y = 110.0f;         GUI.vTb2[2].Z = 0.5f;
        GUI.vTb2[2].RHW = 1.0f;         GUI.vTb2[2].U = 0;      GUI.vTb2[2].V = 1;
        GUI.vTb2[3].X = (float) SCREEN_WIDTH;           GUI.vTb2[3].Y = 110.0f;         GUI.vTb2[3].Z = 0.5f;
        GUI.vTb2[3].RHW = 1.0f;         GUI.vTb2[3].U = 1;      GUI.vTb2[3].V = 1;

        d3ddev->CreateVertexBuffer(4*sizeof(VERTEX_2D), 0, FVF_2D, D3DPOOL_MANAGED, &GUI.bTb2, NULL);
        GUI.bTb2->Lock(0, 0, (void**)&pVoid, 0);
        memcpy(pVoid, GUI.vTb2, 4*sizeof(VERTEX_2D));
        GUI.bTb2->Unlock();
        D3DXCreateTextureFromFileA(d3ddev, "GUI\\Image\\tTb2.png", &GUI.tTb2);

        //*** Tab 3 (environment) ***
        GUI.vTb3[0].X = (float) SCREEN_WIDTH-50;        GUI.vTb3[0].Y = 110.0f;         GUI.vTb3[0].Z = 0.5f;
        GUI.vTb3[0].RHW = 1.0f;         GUI.vTb3[0].U = 0;      GUI.vTb3[0].V = 0;
        GUI.vTb3[1].X = (float) SCREEN_WIDTH;           GUI.vTb3[1].Y = 110.0f;         GUI.vTb3[1].Z = 0.5f;
        GUI.vTb3[1].RHW = 1.0f;         GUI.vTb3[1].U = 1;      GUI.vTb3[1].V = 0;
        GUI.vTb3[2].X = (float) SCREEN_WIDTH-50;        GUI.vTb3[2].Y = 160.0f;         GUI.vTb3[2].Z = 0.5f;
        GUI.vTb3[2].RHW = 1.0f;         GUI.vTb3[2].U = 0;      GUI.vTb3[2].V = 1;
        GUI.vTb3[3].X = (float) SCREEN_WIDTH;           GUI.vTb3[3].Y = 160.0f;         GUI.vTb3[3].Z = 0.5f;
        GUI.vTb3[3].RHW = 1.0f;         GUI.vTb3[3].U = 1;      GUI.vTb3[3].V = 1;

        d3ddev->CreateVertexBuffer(4*sizeof(VERTEX_2D), 0, FVF_2D, D3DPOOL_MANAGED, &GUI.bTb3, NULL);
        GUI.bTb3->Lock(0, 0, (void**)&pVoid, 0);
        memcpy(pVoid, GUI.vTb3, 4*sizeof(VERTEX_2D));
        GUI.bTb3->Unlock();
        D3DXCreateTextureFromFileA(d3ddev, "GUI\\Image\\tTb3.png", &GUI.tTb3);

        //*** Tab 4 (forms) ***
        GUI.vTb4[0].X = (float) SCREEN_WIDTH-50;        GUI.vTb4[0].Y = 160.0f;         GUI.vTb4[0].Z = 0.5f;
        GUI.vTb4[0].RHW = 1.0f;         GUI.vTb4[0].U = 0;      GUI.vTb4[0].V = 0;
        GUI.vTb4[1].X = (float) SCREEN_WIDTH;           GUI.vTb4[1].Y = 160.0f;         GUI.vTb4[1].Z = 0.5f;
        GUI.vTb4[1].RHW = 1.0f;         GUI.vTb4[1].U = 1;      GUI.vTb4[1].V = 0;
        GUI.vTb4[2].X = (float) SCREEN_WIDTH-50;        GUI.vTb4[2].Y = 210.0f;         GUI.vTb4[2].Z = 0.5f;
        GUI.vTb4[2].RHW = 1.0f;         GUI.vTb4[2].U = 0;      GUI.vTb4[2].V = 1;
        GUI.vTb4[3].X = (float) SCREEN_WIDTH;           GUI.vTb4[3].Y = 210.0f;         GUI.vTb4[3].Z = 0.5f;
        GUI.vTb4[3].RHW = 1.0f;         GUI.vTb4[3].U = 1;      GUI.vTb4[3].V = 1;

        d3ddev->CreateVertexBuffer(4*sizeof(VERTEX_2D), 0, FVF_2D, D3DPOOL_MANAGED, &GUI.bTb4, NULL);
        GUI.bTb4->Lock(0, 0, (void**)&pVoid, 0);
        memcpy(pVoid, GUI.vTb4, 4*sizeof(VERTEX_2D));
        GUI.bTb4->Unlock();
        D3DXCreateTextureFromFileA(d3ddev, "GUI\\Image\\tTb4.png", &GUI.tTb4);
}


#endif
```

5.13 global_defines.h

```
#ifndef global_defines
#define global_defines

#include<d3d9.h>
#include<d3dx9.h>

//global defines
int SCREEN_WIDTH = NULL;        //will be retrived in main
int SCREEN_HEIGHT = NULL;       //
char INI_DIR[255];              //

//d3d9 defines for 3d
#define FVF_3D (D3DFVF_XYZ | D3DFVF_NORMAL | D3DFVF_TEX1)
struct VERTEX_3D {FLOAT X, Y, Z; D3DVECTOR NORMAL; FLOAT U, V;};
```

```
//d3d9 defines for 2d
#define FVF_2D (D3DFVF_XYZRHW | D3DFVF_TEX1)
struct VERTEX_2D {FLOAT X, Y, Z, RHW; FLOAT U, V;};

float FPS[5] = {1,1,1,1,1};

#define PI 3.14159265

#endif
```

5.14 win_defines.h

```
#ifndef win_defines
#define win_defines

#include"global_defines.h"

LPCSTR WIN_TIT = "World Beta";

DWORD WIN_STYLE = WS_EX_TOPMOST | WS_POPUP;

#define WM_CHILDMSG             0x0030  //communication menu -> programm

//height map
#define HM_DHM25                0x01
#define HM_DHM200               0x02
#define HM_File                 0x03
#define HM_DIV                  0x10

//texture specification
#define HM_Texture              0x04
#define HM_Texture_flt          0x11
#define TX_GRASS                0x01
#define TX_SNOW                 0x02
#define TX_TXT1                 0x03
#define TX_TXT2                 0x04
#define TX_TXT3                 0x05
#define TX_SPOT                 0x06

//environment
#define EV_Skysphere            0x05
#define EV_CLRCLR               0x06
#define EV_LIGHTCLR             0x07
#define EV_LIGHTDIR             0x08
#define EV_LIGHTSTEEP           0x09

//objects
#define OB_RESTITUTION          0x12
#define OB_FRICTION             0x13
#define OB_RESET                0x14
#define OB_TIME                 0x15

#endif
```

5.15 camera_defines.h

```
#ifndef camera_defines
#define camera_defines

float SENSITIVITY;
float SPEED;
float WHEEL_MULTIP;
float SHIFT_MOV_MULTIP;
float SHIFT_ROT_MULTIP;
float CTRL_MOV_MULTIP;
float CTRL_ROT_MULTIP;

#endif
```

5.16 color_defines.h

```
#ifndef color_defines
#define color_defines

#include<d3d9.h>
#include<d3dx9.h>

D3DCOLOR CLRCOLOR;

D3DCOLOR FONTCOLOR_CONSOLE;
#endif
```

5.17 hmap_defines.h

```
#ifndef hmap_defines
#define hmap_defines

int HMAP_WIDTH;
int HMAP_HEIGHT;
int HMAP_SIZE;

float HMAP_DIV;

#endif
```