Datavetenskap

Conny Ekholm
Christian Olsson

3D-visualisering av planlösningar i Direct3D

Examensarbete

2004:16

3D-visualisering av planlösningar i Direct3D

Conny Ekholm
Christian Olsson

© 2004 Författarna och Karlstads Universitet

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är vårt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Conny Ekholm

Christian Olsson

Godkänd, 20040603

Handledare: Johan Garcia

Examinator: Stefan Lindskog

Sammanfattning

Denna rapport beskriver ett examensarbete på C-nivå i datavetenskap. Rapporten beskriver konstruktionen två program för att visualisera 2D-planlösningar i 3D-miljö. För att utföra detta användes Microsoft's DirectX.

Rapporten tar upp Microsoft's DirectX samt konstruktionsuppbyggnaden av de två program som skapades. Ett program för 3D-motorn samt ett för bildanalys av 2D-planlösning.

Programmet för 3D-motorn konstruerades att läsa in data från en fil som beskriver objekt och objektens placering i en 3D-värld. En användare kan röra sig runt i 3D-världen med hjälp av tangentbordet för att kunna studera planlösningen från olika vyer. 3D-motorn innehåller inte någon kollisionsdektektering vilket medför att en användare kan vandra rakt igenom objekt i 3D-världen.

Bildanalysprogrammet är ett program som är till för att konstruera objekt till en 3D-värld med en 2D-planlösning som underlag. Planlösningen som används visas som en bakgrundsbild i fönstret för användaren. Med hjälp av bakgrundsbilden kan användaren använda ett antal kommandon samt musen för att rita ut objekt. De kommandon som användaren kan använda sig av visas för användaren i en meny. Med hjälp av tangentbordet väljer användaren det kommandot användaren vill använda och markerar därefter ut objektet med musen på planlösningen. Användaren kan sedan spara den värld som skapats till en fil som sedan kan användas av 3D-motorn.

Rapporten tar även upp konstruktionen av filen som objekten sparas i för att användas i 3D-motorn. I filen finns de objekt som användaren konstruerat med bildanalysprogrammet. Objekten beskrivs med en objektstruktur med ett antal variabler.

3D Visualization of Blueprints in Direct3D

This examination report describes a bachelor project in computer science. The examination report describes a program to visualize blueprints for different purposes in a 3D animated world in which a user can move.

The constructions of the two programs that were created to perform the purpose are described in the report which also mentions Microsoft's DirectX. The programs are: one for the 3D engine and one for picture analysis of blueprints.

The program for the 3D engine was constructed in a way that it could read data from a file that describes objects and the objects position in an 3D world. A user can move in a 3D world and look around in different views. The 3D engine does not contain any collision detection and this have the effect that the user can move through objects in the 3D world.

The picture analysis program makes it possible to construct objects to a 3D engine from a blueprint. The blueprint which are used will be shown as a background image in the users window. With the help of the background picture the user can use a number of commands and also the mouse to draw objects. Those commands that the user can use are shown in a menu. With the help of the keyboard the user can choose a command and then the user can mark the place on the blueprint where the object should be. The user can then save the world that has been created to a file which later can be used by the 3D engine.

The report also describes the construction of the file in which the objects are saved in, to be used in the 3D engine. The file contains the objects that the user has constructed with the picture analysis program. The objects describes in a object structure with a number of variables.

Innehåll

1 Inledning **1**

2 DirectX **3**

2.1 Vad är DirectX . . . 3

2.2 Direct3D . . . 4

3 3D-motorn **6**

3.1 Kod för initiering av fönsterhantering i Windows . . . 6

3.2 Kod för initiering av DirectX . . . 7

3.3 Konstruktion . . . 7

3.3.1 Inledningen av konstruktionen . . . 8

3.3.2 De första grafiska testerna . . . 8

3.3.3 Koordinatsystemet i 3D-världen . . . 11

3.3.4 Funktioner för uppritande av objekt i 3D-världen . . . 12

3.3.5 Camera . . . 14

3.3.6 Light . . . 16

3.4 Vidareutveckling i 3D-motorn . . . 16

4 Filhantering **20**

4.1 Inledning . . . 20

4.2 Objektstrukturen . . . 20

4.3 Skriva och läsa från filen . . . 22

4.4 Problem med strukturen . . . 22

5 Bildtolkning **24**

5.1 Musimplementation . . . 24

5.2 Bildanalysprogrammet . . . 25

5.2.1 Meny . . . 25
5.2.2 Uppritning av planlösning och skalning . . . 28
5.2.3 Texturlösning och koordinattolkning av mus . . . 29
5.2.4 Utritning av 2D objekt på planlösning . . . 31
5.2.5 Omvandling av 2D till 3D . . . 31
5.2.6 Multiplicering av textur över objekt . . . 32
5.2.7 Skapa fönster . . . 33
5.3 Vidareutveckling i bildanalysprogrammet . . . 33

6 Erfarenheter **36**

7 Slutsats **37**

Referenser **39**

A Bilaga A - Funktionsbeskrivning **40**
A.1 Funktioner från 3D-motorn . . . 40
A.1.1 drawScene . . . 40
A.1.2 WinMain . . . 40
A.1.3 direct3dInit . . . 41
A.1.4 setupView . . . 41
A.1.5 setupTexture . . . 41
A.1.6 release . . . 42
A.1.7 releaseVB . . . 42
A.1.8 setupRenderState . . . 42
A.1.9 setMaterial . . . 42
A.1.10 setLight . . . 43
A.1.11 create . . . 43
A.1.12 createWindow . . . 44

A.1.13 wndProc 44
A.1.14 GetHwnd 45
A.1.15 drawCube 45
A.1.16 drawQuad 46
A.1.17 drawWorld 47
A.1.18 fromFile 47
A.2 Funktioner från bildanalysprogrammet 47
A.2.1 saveTofile 47
A.2.2 picProgram 48
A.2.3 drawSketch 48
A.2.4 menu 49
A.2.5 initFont 49
A.2.6 getTexNr 49
A.2.7 defineWallExactly 49
A.2.8 createDoor 50
A.2.9 splitWall2 50
A.2.10 createWindow 51
A.2.11 splitWall 51
A.2.12 save 51
A.3 Musfunktioner 52
A.3.1 initMouse 52
A.3.2 update 52
A.3.3 drawCursor 52
A.3.4 mouseButtondown 53
A.3.5 getMousePos 53
A.3.6 getMousePos3 53

Figurer

3.1 Fönsterlist och muspekare 6
3.2 Kod för utritning av objekt. 9
3.3 Rektangel 10
3.4 Kub 11
3.5 Definiering av konstant för egenskaper för noder. 13
3.6 Skapande av en vertexbuffer. 13
3.7 Kod för att låsa en vertexbuffer. 14
3.8 Tilldelning av nod till vertexbuffer. 14
3.9 Kod för att låsa upp vertexbuffer. 14
3.10 Vinkeln teta samt position och fokuseringspunkt. 15
3.11 Ljussättning mot en kubs olika sidor. 17
3.12 En vy i 3D-motorn inne i en lägenhet. 18
4.1 Objektstrukt som används för att lagra objekt till fil. 21
4.2 Skriver objekt till fil. 22
4.3 Läser in objekt från fil. 23
5.1 Menyns utseende vid programstart. 26
5.2 Menyns utseende för väggfunktioner. 26
5.3 Menyns utseende för texturval. 28
5.4 Över en svart bakgrund ser hårkorset vitt ut. 30
5.5 Över en vit bakgrund ser hårkorset svart ut. 30
5.6 Över brytningen av en delad svartvit yta. 31
5.7 Ett fönster i 3D-världen. 34
5.8 Vy av bildanalysprogrammet. 34

1 Inledning

Projektet går ut på att konstruera program för att visualisera 2D planlösningar i en 3D-miljö. En användare kan läsa in sin 2D planlösning i programmet som en bild, och med hjälp av programmet skapa en modell av planlösningen. Denna modell kan sedan användas för att generera en 3D-planlösning där användaren kan röra sig runt med hjälp av programmet. Detta för att se hur olika rum, byggnader eller liknande objekt kommer att se ut i den riktiga världen.

Projektet är tänkt att användas av till exempel mäklare eller arkitekter. En arkitekt kan använda sig av programmet för att testa deras ideér om konstruktioner och se hur dessa blir i en 3D miljö. En mäklare kan använda sig av programmet för att visa tänkta kunder hur en lägenhet eller lokal ser ut utan att kunden behöver besöka platsen. Det är även tänkt att användare oberoende av yrke eller syfte kan använda sig av programmet för att testa olika ideér. Ett syfte som var tänkt med detta projektet var att vi, författarna till rapporten, skulle få en större kunskap om programmering i DirectX med Direct3D. I denna rapport kommer därför inte andra grafikbiblotek tas upp som till exempel OpenGL [1] eller andra möjligheter att lösa visualiseringen, till exempel andra 3D-motorer.

Projektet delas upp i två större delar. Den första delen är 3D-motorn som skall användas för att rita upp världen som användaren vill använda sig av. 3D-motorn skall läsa in objektbeskrivningar från en fil. Till att rita upp objekten i 3D använder 3D-motorn sig av Microsoft DirectX's 3D-bibliotek, Direct3D. I denna 3D-motor skall användaren kunna röra sig runt och studera 3D-världen i olika vinklar. Detta för att erhålla en känsla av världen som inte kan erhållas med en 2D-planlösning.

Den andra stora delen av projektet är ett program för att läsa in 2D-planlösningar och omvandla denna till en fil innehållande objekt som kan tolkas av 3D-motorn. Programmet är gjort så att användaren får 2D-planlösningen som en mall som användaren kan använda sig av för att rita in de olika objekten i världen. Dessa två program kommunicerar mellan varandra via en fil med gemensamt format.

Denna rapport är indelad i ett antal kapitel. Först beskrivs DirectX, därefter kommer utvecklingen av 3D-motorn att beskrivas i ett kapitel. Detta följs av en beskrivning av filformatet som används för att kommunicera mellan 3D-motorn och bildanalysprogrammet. Efter kapitlet där filen beskrivs kommer utvecklingen av bildanalysprogrammet att beskrivas.

2 DirectX

I följande kapitel beskrivs DirectX och Direct3D, historik och vad de är samt vad de kan användas till.

2.1 Vad är DirectX

Microsoft skapade den första versionen av DirectX 1995, som då kallades *the game SDK*, för att locka spelprogrammerare bort från MS-DOS och till Microsoft Windows. Före DirectX hade Microsoft Windows begränsade möjligheter när det kom till direkt åtkomst av grafik och ljud, och det var ofta för långsamt för att användas till spel. Microsoft insåg att de flesta spel behövde direkt åtkomst till hårdvaran för att kunna exekveras snabbt och skapade därför DirectX så att Microsoft Windows-programmerare kunde få direkt åtkomst till hårdvaran. Men detta skulle kunna uppnås utan hårdvaruspecifik kod. Det är därför som Microsoft slutligen döpte om *the Game SDK* till DirectX. Nu är DirectX en standard för Microsoft Windows för att spela upp ljud, grafik och multimedia och är inbyggt i operativsystemet. DirectX är plattformsberoende så det fungerar bara i Microsoft Windows operativsystem och inte i andra plattformar.

DirectX har en stor fördel, den gör så att all hårdvara ser likadan ut för programmerarens oavsett viket typ av datachip som sitter på grafikkortet eller vilken tillverkare som konstruerat kortet. Alla kort av en viss typ, till exempel grafikkort, ser likadana ut från programmerarens vy via DirectX. DirectX erhåller detta oberoende till hårdvaran genom att använda drivrutinerna, vilket tar DirectX-kommandon och konverterar de till lämpliga hårdvarukommandon.

DirectGraphics Den här delen av DirectX tar hand om uppritning av grafik på skärmen. I tidigare versioner av DirectX var det separata komponenter för 2D grafik (kallad DirectDraw) och 3D grafik (kallad Direct3D). I DirectX 8 så har dessa komponenter blivit kombinerade till DirectGraphics. Dock ska tilläggas att DirectDraw och

Direct3D initieras separat.

DirectAudio Den här delen av DirectX är till för att göra musik och ljudeffekter. Den används för att kunna spela upp ljudeffekter och ljudfiler som låter som om de kom från en punkt i en 3D värld, komplett med specialfiltreringseffekter.

DirectPlay Den här delen används för nätverksprogrammering. Med den kan multiplayerspel skapas som lätt fungerar över internet, modem och LAN. Det har även stöd för realtidsröstkommunikation i multiplayerspel.

DirectInput Den här delen skapar access till tangentbord, möss, joysticks och andra indataenheter. Den här delen gör det även möjligt att kontrollera Force Feedback-enhet.

DirectShow Den här delen gör det möjligt att spela upp video och multimediasekvenser.

2.2 Direct3D

Direct3D är integrerat i DirectGraphics som är en del i DirectX. Direct3D används för att utveckla interaktiva realtids 3D applikationer. För dessa applikationer har Direct3D följande funktionalitet enligt [4]:

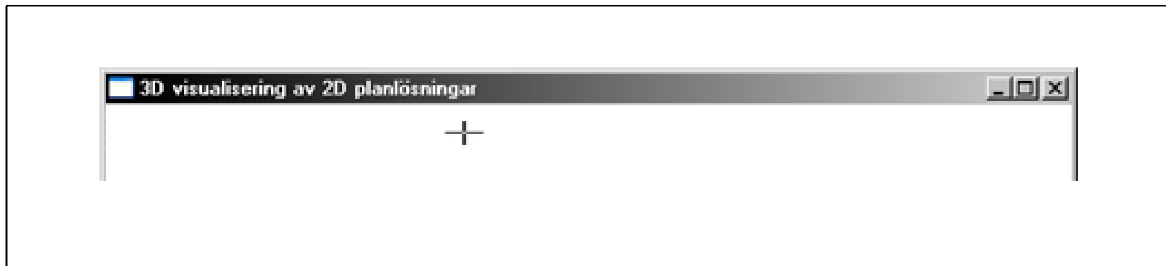
Hårdvaruoberoende - Direct3D använder drivrutiner vilket medför att applikationer kan fungera oberoende av hårdvara.

Drivrutiner har standardfunktionalitet - Garanterar applikationerna att drivrutinerna som stödjer Direct3D alltid tillhandahåller ett definierat minimum av funktionalitet. Applikationer som använder dessa funktioner kan alltså vara säkra på att programmet fungerar oberoende av hårdvara. Direct3D har en specifikation som hårdvarutillverkare kan använda sig av för att tillverka ett grafikkort som stödjer Direct3D. Applikationer som sedan använder sig av sådana hårdvaruspecifika funktioner får bättre prestanda.

Hjälper applikationer att implementera 3D - Direct3D är ett grafikbibliotek som

tillhandahåller funktioner för 3D grafik. Applikationer som kräver 3D grafik kan skapas mycket enklare med hjälp av Direct3D istället för att skriva alla funktioner själv.

Använda mjukvara istället för grafikkortet - en viktig egenskap hos Direct3D är att om inte hårdvaran har en viss funktionalitet så tillhandahålls denna funktionalitet i mjukvaran. Dock så är funktionallitet som tillhandahålls från hårdvaran mycket snabbare än om den tillhandahålls från mjukvaran.



Figur 3.1: Fönsterlist och muspekare

3 3D-motorn

I detta kapitel beskrivs konstruktionen av 3D-motorn och delar runt den. Kapitlet har delats in i ett antal avsnitt. Först kommer initieringen av både Windows samt Direct3D att beskrivas. Detta följs av ett avsnitt som tar upp konstruktionen. Detta avsnitt delades in i ett antal delavsnitt som beskriver delar av konstruktionen. I detta kapitel kommer det även att nämnas en del funktioner med funktionsnamn som används till olika delar av 3D-motorn, för en mera fullständig förklaring av funktionerna så kommer dessa att ta upp enskilt i bilaga A.

3.1 Kod för initiering av fönsterhantering i Windows

Till en början måste Microsoft Windows grafikbibliotek, windows.h, inkluderas. Detta görs för att kunna använda olika funktioner som till exempel att skapa ett nytt fönster i Microsoft Windows. Här används inte vanliga main() utan istället används WinMain(). Denna funktion tar in lite andra parametrar än vad den vanliga main-funktionen gör. Att skapa ett fönster är en tvåstegs process.

Först definieras något som kallas för *window class*. Den här window klassen som skapats är till för att definiera egenskaper för fönstret. Exempel på egenskaper är rubrik i fönsterlisten och typ av muspekare. I figur 3.1 visas fönsterlisten och muspekaren. Funktionen RegisterClassEx ska sedan anropas med denna *window class* som parameter.

Nästa steg är att skapa ett fönster och länka det till windowsklassen. För att skapa ett fönster anropas funktionen CreateWindowEx. Exempel på parametrar här är höjd och bredd på fönstret samt en pekare till windowklassen.

När Winmain returnerar så avslutas fönstret. Detta medför att en loop får användas i Winmainfunktionen. Denna loop får villkoret 1 (true) i sig för att inte Winmain loopen skall exekvera klar och returnera. I Winmain placerades något som på engelska kallas för *message pump* och som översatt betyder meddelandepump. Denna pump kotrollerar om Microsoft Windows lämnat meddelanden, till exempel om en tangent har tryckts ned på tangentbordet eller att musen har använts.

3.2 Kod för initiering av DirectX

Det första som görs är att anropa funktionen Direct3DCreate8 från klassen IDirect3D8. Den tar ett argument och det skall alltid vara satt till D3D_SDK_VERSION vilket ser till att headerfilerna är i sync med den använda versionen DirectX. Nästa steg är att skapa en device. Detta görs genom att anropa funktionen CreateDevice. I funktion CreateDevice anges egenskaper som upplösning, om uppritningen skall vara i fullskärm eller i ett fönster, etc. Denna rapport går inte så djupt in på detta ämne då det behandlas i böcker som tar upp Direct3D, till exempel [2].

3.3 Konstruktion

I detta avsnitt om konstruktionen av 3D-motorn beskrivs de olika stegen och delarna som skapats och genomförts för konstruktionen av 3D-motorn. För att få en mera detaljerad beskrivning av funktioner som tas upp i följande kapitel hänvisas till bilaga A.

3.3.1 Inledningen av konstruktionen

Det första som gjordes i 3D-motorn var att initiera ett fönster i Windows som användes för att rita upp 3D-effekter i. Efter initieringen av Windows initierades Direct3D som användes i detta projekt för att rita ut grafiken på skärmen. När dessa två initieringssteg var avklarade kunde ett fönster med en vit bakgrundsfärg ses. Färgen på bakgrunden valdes till vit men den kan även väljas till andra färger. För att välja färgen används RGB-färger (R = Röd, G = Grön, B = Blå). Ett värde mellan 0 till 255 sätts för varje färg. Den funktion som hanterar uppritningen på skärmen är döpt till drawScene och det är i denna funktion som bakgrundsfärgen sattes.

3.3.2 De första grafiska testerna

Efter initieringarna av både Windows och DirectX började tester med att rita upp ett antal grafiska objekt. I detta delavsnitt beskrivs de första testerna inom detta område. För att rita upp en polygon på skärmen bör följande steg utföras enligt [2] som illustreras av koden i figur 3.2:

1. Töm *backbuffern* genom att anropa IDirect3DDevice8s Clear funktion. I denna funktion sättes även bakgrundsfärgen.

2. Anropa BeginScene för att indikera för Direct3D att börja rendera en scen.

3. Sätt källan för dataadressen till den vertexbuffer (vertexbuffer beskrivs i delavsnitt 3.3.4) du vill rendera. Detta görs med ett anrop till IDirect3DDevice8s funktion SetStreamSource.

4. Ange din vertex shader genom att anropa SetVertexShader. En vertex shader talar om för Direct3D vad för data dina noder innehåller. Exempel på data som en nod kan innehålla är x-, y-, z-position, färg och textur. I detta projekt definierades en klass för dessa noder som kallas CVertex. Vertex är det engelska ordet för nod. Här skickas en

```
directx.dxDev->Clear(0, NULL, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER,
                     D3DCOLOR_XRGB(255, 255, 255), 1, 0);
directx.dxDev->BeginScene();

directx.dxDev->SetStreamSource(0,directx.dxVB[vbnr],sizeof(CVertex));
directx.dxDev->SetVertexShader(D3DFVF_CVERTEX);
directx.dxDev->SetTexture(0,directx.dxTex[texnr]);
directx.dxDev->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 12);
directx.dxDev->EndScene();
directx.dxDev->Present(NULL,NULL,NULL,NULL);
```

Figur 3.2: Kod för utritning av objekt.

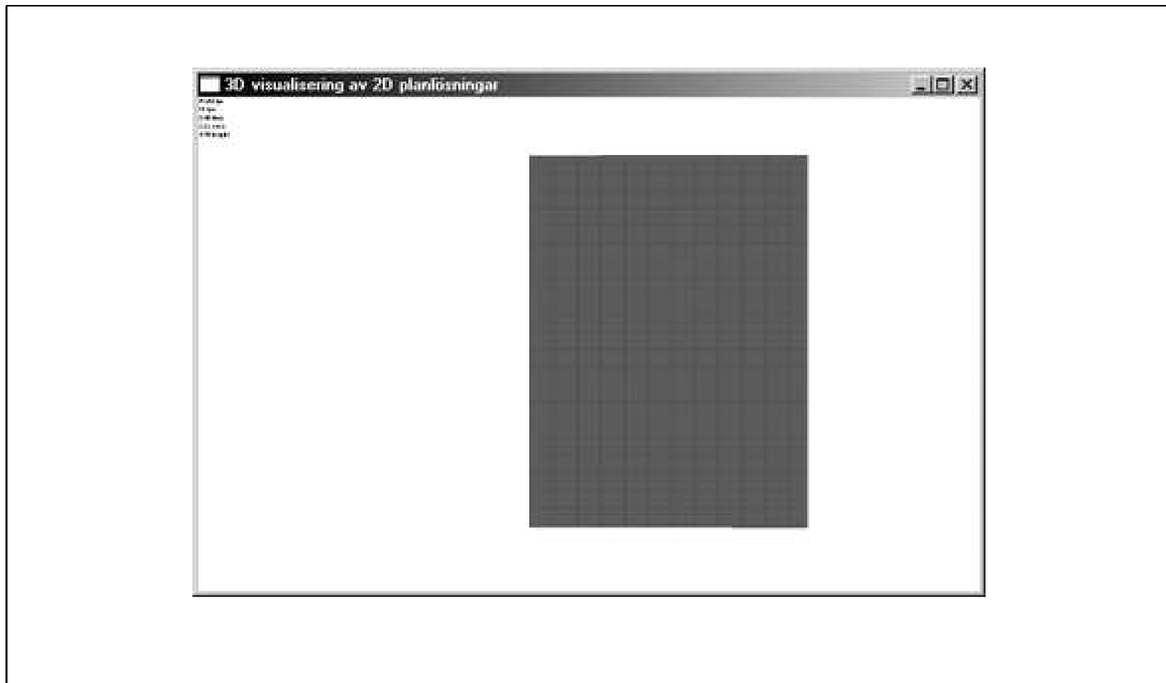
fördefinierad konstant som heter D3DFVF_CVERTEX in. Denna konstant beskriver vad för data noden innehåller.

5. Om en textur skall användas på polygonen så setts den med funktionen SetTexture. En textur är ytan på objektet, det vill säga att lägga olika bilder för att simulera mönster och utseende på objektet. Anropa DrawPrimitive för att rendera noderna med angiven data och vertex shader. Detta sköter själva uppritningen av objektet på skärmen.

6. Anropa EndScene för att låta Direct3D veta att du är klar med att rendera din scen.

7. Presentera din Backbuffergenom att anropa Present.

Här gås det närmare in på det som kan tyckas verka oklart av koden i figur 3.2. Makrot D3DCOLOR_XRGB(R, G, B) används för att specificera en färg. Den tar in parametrarna R,G,B (röd, grön, blå), där 0,0,0 motsvarar svart och 255,255,255 motsvarar vit färg. I funktionen SetStreamSource så skickas dxVB[] med som är en array med pekare till vertexbuffern där polygonens koordinater ligger sparade. Utförligare förklaring av vertexbuffer kommer i kap 3.3.4 men kan kort säga att det är ett minne där polygonens noder



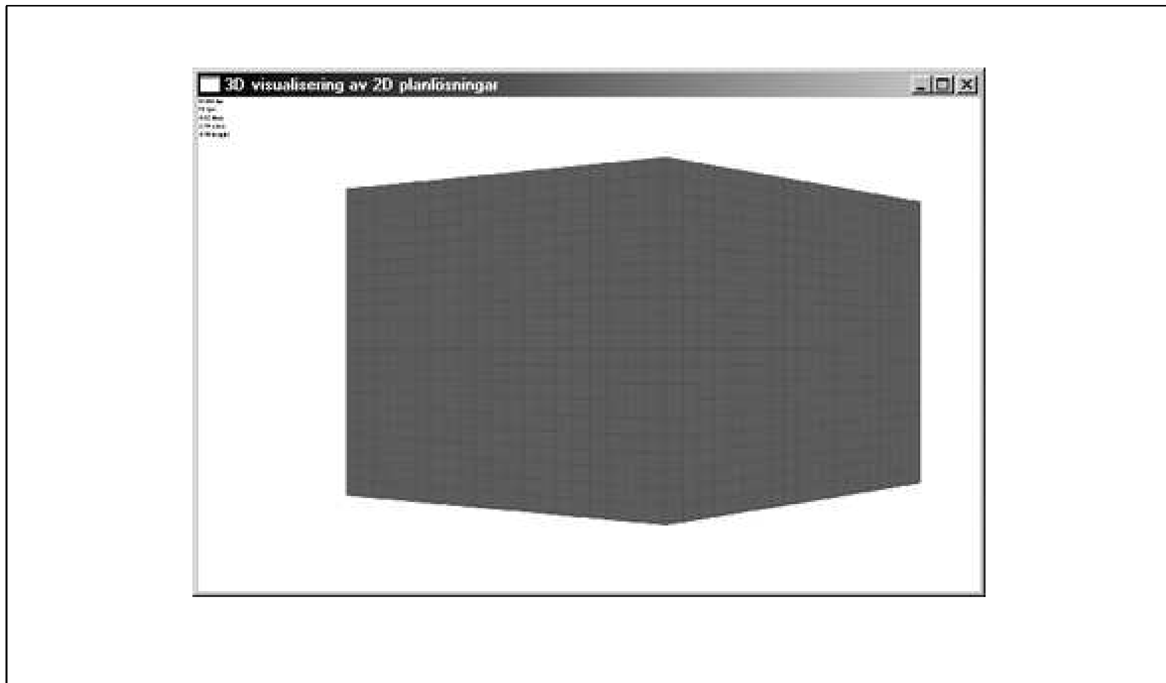
Figur 3.3: Rektangel

sparas.

Vertexbuffern definieras i funktionerna drawCube och drawQuad vilka beskrives i detalj senare. I funktionen SetTexture skickas en pekare till en textur. Arrayen dxTex innehåller pekare till olika texturer. Genom att anropa med rätt indexering fås den specifika textur som önskats.

Drawprimitives första argument är D3DPT_TRIANGLELIST vilket innebär att noderna skall ritas upp med triangellistor. I dessa triangellistor anges tre stycken noder för att definiera en triangel. Förutom dessa triangellistor finns andra alternativ att använda sig av. Ett av dessa är så kallade triangelstrips, vars funktion är att rita flera olika trianglar. Sedan utgår det ifrån de två senaste noderna för att ange nästkommande triangels första noder. På så vis optimeras renderingen.

Under de första testerna att rita upp grafik provades objekten att rotera samt att förflytta dessa i olika riktningar. I figurerna 3.3 och 3.4 visas en rektangel samt en kub.



Figur 3.4: Kub

Detta var de två första objekten som testades att rita upp i 3D motorn.

3.3.3 Koordinatsystemet i 3D-världen

I detta delavsnitt beskrivs koordinatsystemet som används i 3D-världen samt vissa delar runt detta.

3D-världen i DirectX har ett koordinatsystem där Y-axeln anger positionering i höjdled med ökande värden i riktning uppåt. X-axeln anger positionering i horisontalled med ökande värden i höger riktning och den sista axeln är Z-axeln som anger positionering i djupled med ökande värden in mot skärmen från användarens synvinkel.

DirectX använder sig av matriser för att visualisera ett objekt i dess rymd. Det finns tre grundmatriser: world-, view- och projectmatris.

World-matrisen används för att positionera ut objekt i 3D-världen.

View-matrisen används för att ange var i rymden användaren skall vara positionerad och

vart denna ska titta. View-matrisen har tre stycken egenskaper som skall sättas. Den första egenskapen är en tredimensionell vektor som anger x-, y-, z-position för vart användaren är placerad i 3D-världen. Den andra egenskapen är ytterligare en 3D vektor som anger vilken riktning som skall definieras som uppåt. Alltså uppåt i användarens synvinkel är uppåt i 3D-världen. Den sista egenskapen är en 3D-vektor som anger positionen som användaren tittar på.

Sista matrisen, projectmatrisen, används för att definiera spridningen av synfältet samt hur långt synfältet skall vara, det vill säga hur mycket som skall ritas upp av världen på skärmen som användaren skall se. Denna kan jämföras med hur långt användaren kan se i den riktiga världen. Om det, till exempel, är en klar dag är sikten klar långt bort till horisonten. Däremot på en dimmig dag är sikten inte till horisonten, utan kortare.

3.3.4 Funktioner för uppritande av objekt i 3D-världen

I detta delavsnitt beskrivs de två funktioner som skapades för att rita upp objekt som skulle projiceras i 3D-världen.

Två funktioner skapades, en som beskriver en kub, drawCube, och den andra funktionen, drawQuad, som beskriver en fyrkant. Funktionerna tar in parametrar som behövs för att beskriva en kub eller en fyrkant.

Funktionerna har definierat punkter i 3D-världen och dessa punkter skall sparas i vertexbuffer, vilket nämndes kort tidigare i kapitel 3.3.2. En vertexbuffer är en bit minne, antingen RAM eller grafikminne. Varje vertexbuffer kan peka till en viss mängd data. En sådan här vertexbuffer läts peka till alla koordinater för en kub eller en fyrkant (beroende på funktion). Eftersom ett antal olika objekt kommer vara uppritade på skärmen har en array av vertexbuffrar skapats. För att skapa en vertexbuffer så används klassen IDirect3DDevice8s CreateVertexBuffer funktion.

I figur 3.5 visas vår egendefinierade konstant som beskriver formatet på vertexstrukturen, det vill säga egenskaperna som varje nod i polygonen har. Konstanten D3DFVF_XYZ

```
const unsigned long D3DFVF_CVERTEX = D3DFVF_XYZ | D3DFVF_NORMAL |
                    D3DFVF_TEX1;
```

Figur 3.5: Definiering av konstant för egenskaper för noder.

```
directx.dxDev->CreateVertexBuffer(sizeof(CVertex)*36, 0,
        D3DFVF_CVERTEX, D3DPOOL_MANAGED, &directx.dxVB[vbnr]);
```

Figur 3.6: Skapande av en vertexbuffer.

innebär att en punkt i 3D-världen definierats med x-, y- och z-koordinater. Konstanten D3DFVF_NORMAL visar att varje nod har definierat normalen mot ytan. Detta används för ljussättningen. Konstanten D3DFVF_TEX1 visar att polygonen kommer att textureras. I nästa steg anropades CreateVertexBuffer för att skapa en vertexbuffer. Detta anrop (i drawCube funktionen) visas i figur 3.6.

CVertex är klassen som definierar vertexstrukturen. Första parametern anger storleken på CVertex klassen multiplicerat med 36, vilket är antalet noder i den kub som skapas i funktionen drawCube. Nästa parameter anger vad vertexbuffern kommer att användas som. I tredje parametern skickas vår fördefinierade konstant med. Parametern D3DPOOL_MANAGED i CreateVertexBuffer säger till Direct3D att hantera denna vertexbuffer oavsett om den är i systemminnet eller grafikminnet. Direct3D flyttar saker från och till grafikminnet när det behövs. Den sista parametern är adressen till pekaren till vertexbuffern.

För att fylla en vertexbuffer utförs tre steg. Först låses vertexbuffern, därefter fylls den och till sist låses den upp. Utdrag ur kod visas ifigur 3.7 för att illustrera.

Här efter körs en for-loop där egenskaperna för en nod (vertex) i taget sätts. Funktionen

```
directx.dxVB[vbnr]->Lock(0,0, (BYTE**)\&vertex,0);
```

Figur 3.7: Kod för att låsa en vertexbuffer.

```
vertex[i].create(x,y,z,nx,ny,nz,u,v);
```

Figur 3.8: Tilldelning av nod till vertexbuffer.

create, som är en egendefinierad funktion som sätter egenskaperna på noden, anropas. I figur 3.8 visas anropet till denna funktion. Notera att hela koden inte visas här utan den finns i bifogad kod. Kod för att låsa upp visas i figur 3.9.

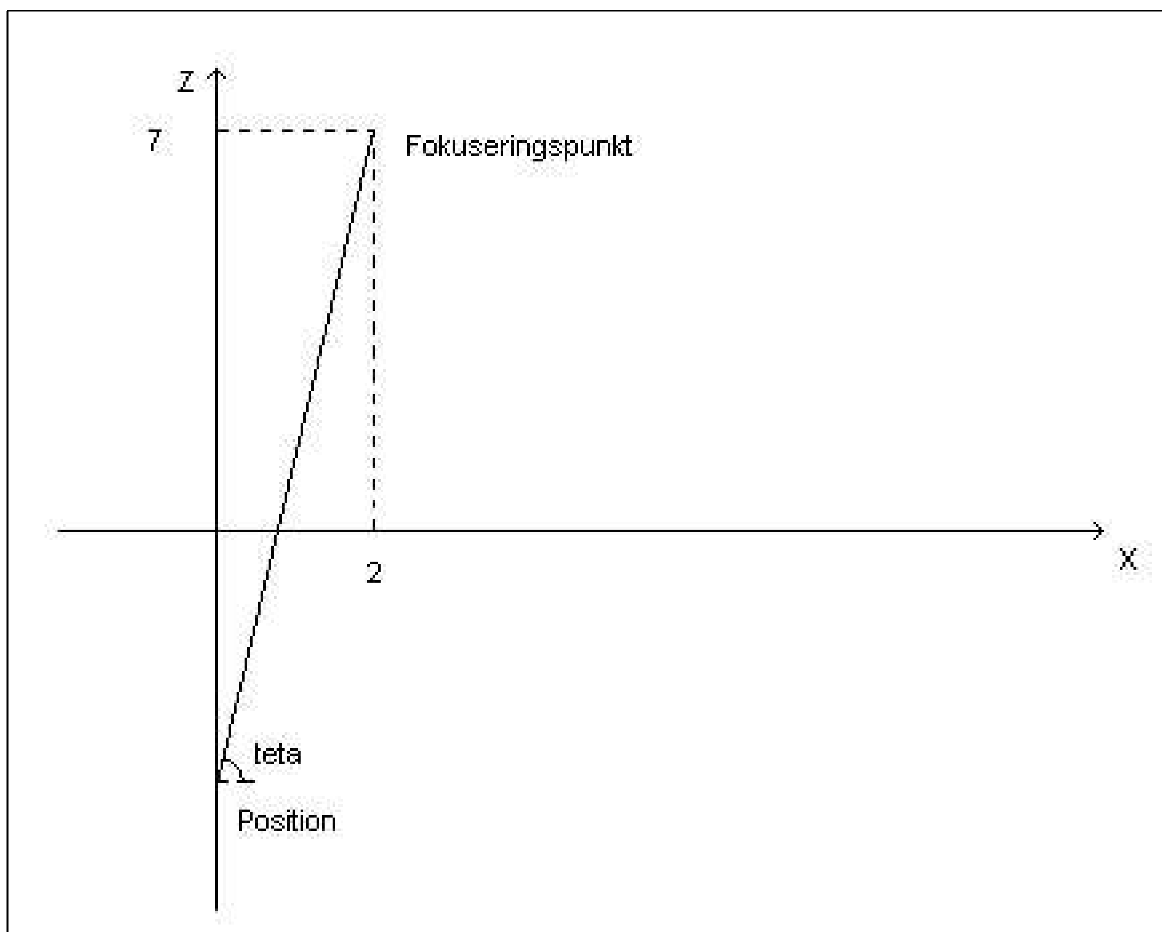
Vad som gjordes härnäst var att skapa funktioner för att läsa in egenskaper för noden från en fil till en strukt. Strukten används sedan för att anropa funktionerna drawCube eller drawQuad. Vilken funktion den anropar beror på värdet i struktens variabel deep. Om variabeln deep är satt till noll så anropas funktionen drawQuad eftersom då är det en fyrkant och ingen kub som skall ritas upp.

3.3.5 Camera

I projektet skulle användaren ha möjligheten att röra oss runt i den 3D-värld som skapats. Koden för att göra detta lades i filen camera och funktionen moveAround. För att ändra positionen användaren står samt tittar på görs förändringar i matrisen viewmat. Först

```
directx.dxVB[vbnr]->Unlock();
```

Figur 3.9: Kod för att låsa upp vertexbuffer.



Figur 3.10: Vinkeln teta samt position och fokuseringspunkt.

definieras konstanter för position och fokuseringspunkt och initieras till lämpliga värden. Användaren står på position 0 i x- och y-axeln och en bit bakåt i z-axeln ifrån dess nollvärde. Användaren tittar framåt (z-led representerar framåt här) och lite snett åt höger. I koden kallas fokuseringspositionerna för x-, y-, z-kamera. I figur 3.10 visas en illustration för ett förtydligande.

Figur 3.10 visas vinkeln teta. Denna vinkel indikerar hur stor vinkeln är från x-axeln till den position användaren tittar åt. Vinkeln teta räknades ut från positionen på fokuseringspunkten (x=2, z=7, y=0).

Funktionen moveAround kontrollerar om någon av tangenterna 'w', 's', 'a' och 'd' har

tryckts ned. Beroende på vilken tangent som tryckts ned bestäms den nya positionen och fokuseringspunkten med hjälp av cosinus och sinus beräkningar. Funktionen kontrollerar även om någon av piltangenterna trycks ned. Om en piltangent tryckts ned så utförs motsvarande kod för ändring av fokuseringspunkt i 3D-världen. Med andra ord så rör användaren sig runt i rummet med 'w', 's', 'a' och 'd' tangenterna och tittar runt med piltangenterna. Dessa ändringar som beräknas fram sparas i variablerna till vår matris viewmat och säger till Direct3D att rensa de gamla värdena för position och synfält och sätta de nya värden som finns lagrat i vår matris viewmat.

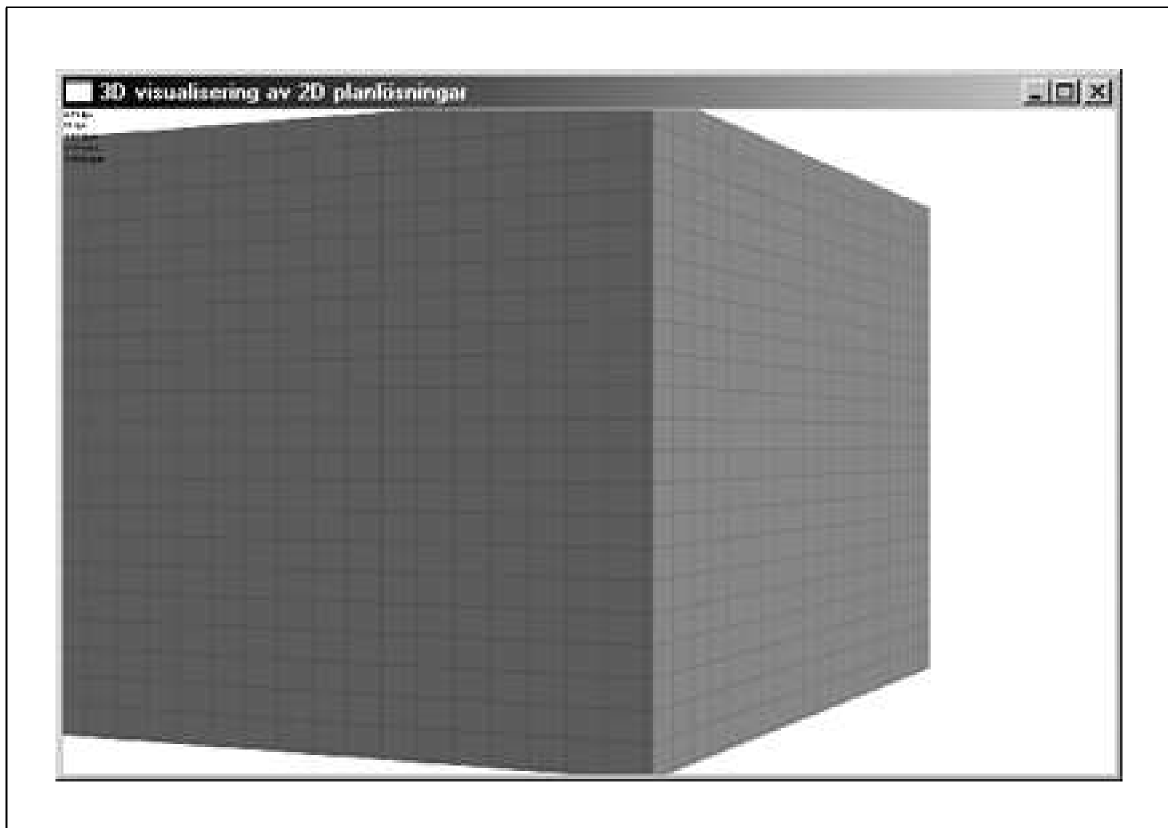
3.3.6 Light

För att 3D-världen skulle se lite bättre ut lades ljussättning till. Med klassen D3DLIGHT8 sattes egenskaperna för ljuset. Det finns tre olika typer av ljus vilka är directional light, pointlight och spotlight. Directional light är ett ljus som sträcker sig oändligt långt och som lyser i en viss riktning. Pointlight är precis som en glödlampa positionerad i rymden. Den lyser upp en ändligt rymd. Spotlight har en position och en riktning och lyser endast upp den punkt där den träffar och inte oändligt långt. Tänk på den som vanliga spotlight som finns på nattklubbar osv. Directional light kräver minst datorkraft. De två andra ljusen kräver ganska mycket datorkraft så de ska användas med måtta. I figur 3.11 visas en ljussatt kub.

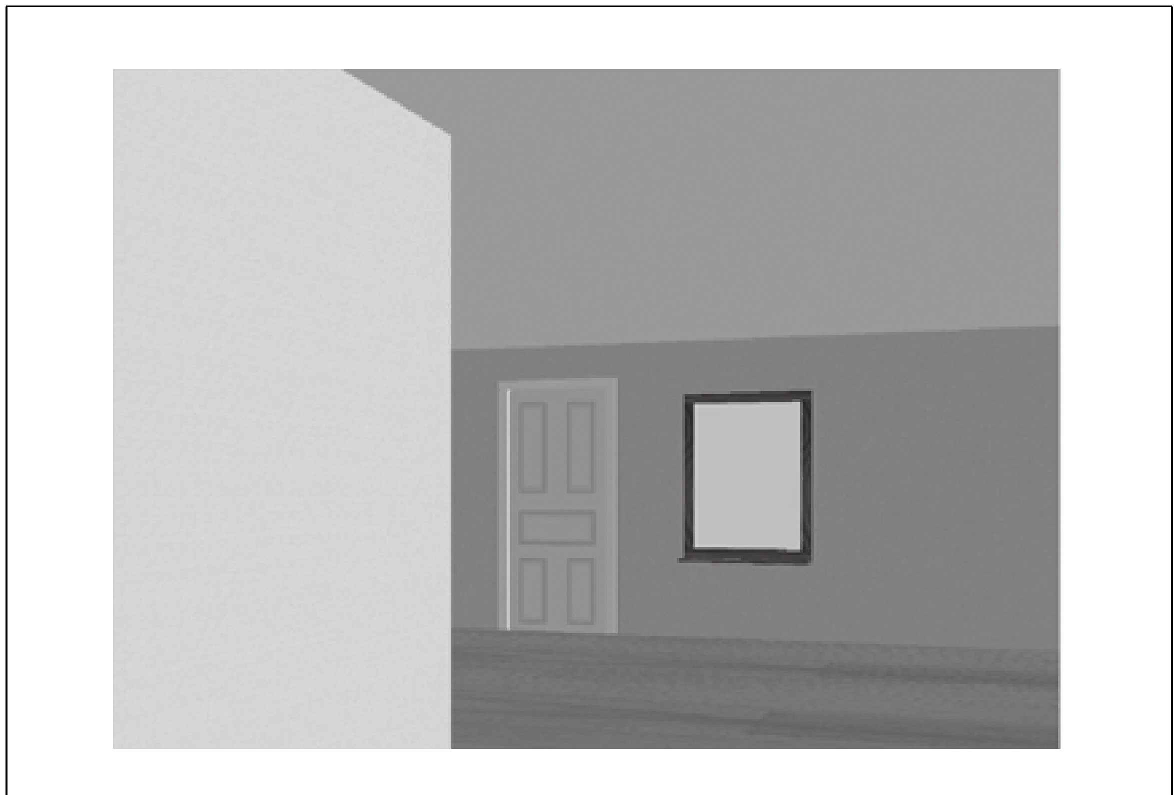
3.4 Vidareutveckling i 3D-motorn

Vid detta tillfälle i projektet kan en vy i 3D-motorn se ut enligt figur 3.12, men då rapporten skrivs har tid ej funnits till att göra de extrasaker som kunde ha varit med i programmet. Ett sådant exempel är kollisionsdetektering, vilket innebär att när användaren kommer fram till en vägg eller annat objekt så kan inte användaren gå igenom objekten.

Ytterligare en sak som kunde förbättras är ljussättningen på fyrkanter (väggar utan tjocklek). När en sida belyses med ljus så belyses även motsatt sida på fyrkanten upp.



Figur 3.11: Ljussättning mot en kubs olika sidor.



Figur 3.12: En vy i 3D-motorn inne i en lägenhet.

Detta problem uppstår inte då kuber används.

Optimering av upppritningshastighet har hela tiden eftersträvats, men självklart finns det ytterligare möjligheter att förbättra detta. En sak som kan förbättras är att göra algoritmer som ser till att väggar som användaren ej kan se inte ritas upp.

Vidare så är programmet beroende av hårdvaran med avseende på hur snabbt rörelser i 3D-världen inträffar. Detta innebär att om olika snabba datorer avänds kan rörelser i 3D-världen gå olika snabbt. Detta borde programmeras så att ett oberoende av hårdvarans hastighet uppnås.

4 Filhantering

Detta kapitlet tar upp överföringen av data mellan 3D-motorn som togs upp i det förra kapitlet och bildanalysprogrammet som tas upp i nästa kapitel.

4.1 Inledning

Filhanteringen till de programmen som konstruerats i projektet, 3D-motorn och bildanalysprogrammet, är till för att kommunicera mellan dessa två programmen och överföra information om hur objekten skall se ut och var dessa skall placeras. För att kommunicera mellan programmen valdes det att användas en fil, detta för att programmen skulle kunna agera enskilt från varandra. Ett annat mål var att kunna spara undan de världar som genererats för att användas till 3D-motorn vid flera tillfällen. En annan fördel med att använda sig av ett så markant sätt att skärma av mellan programmen är att det blir enklare om någon annan vill utveckla ett annat program till ett av de framtagna programmen i projektet.

4.2 Objektstrukturen

Strukturen på filen som används för att föra över information är uppbyggd så att filen lagrar strukter av typen objektstruct som är definierade enligt figur 4.1. Objektstrukt består av de variabler som behövs för att beskriva ett objekt, vbnr, u, v, x-, y-, z-koordinaterna, höjd, bredd och djup, texturnummer samt vilken typ av objekt det är.

Den första variabeln vbnr är ett nummer som används av 3D-motor programmet för att hålla koll på vilket objekt det är. Detta nummer genereras på det sättet att första objektet som skrivs in i filen får nummer noll, därefter ökas numret till ett för det andra objektet och så vidare för efterföljande objekt.

Variablerna u och v är egenskaper för texturen på objektet. När texturen sätts på ett objekt kan det väljas att använda en bild som dras ut över hela objektets storlek så

```
typedef struct objectstruct{
      int vbnr;
      float u;
      float v;
      float x;
      float y;
      float z;
      float width;
      float height;
      float deep;
      int texnr;
      int type;
}objectstruct;
```

Figur 4.1: Objektstrukt som används för att lagra objekt till fil.

att bilden passar till objektet. Detta kan vara lätt att använda med en enfärgad textur utan mönster. Däremot om en textur med möster används kan det vara dumt att dra ut texturen över hela objektets yta. Detta kan medföra att mönstret missformas och inte blir det mönster som öndskats. För att motverka att en textur blir missformas kan flera texturbilder placeras brevid varandra på objektets yta. Då används variablerna u och v. Variabeln u beskriver antalet texturbilders upprepningar över objektet i x-led och v beskriver antalet texturbilders upprepningar över objektet i y-led.

Width, height och deep är variabler för bredd, höjd och djup på objektet. Dessa variabler utgår ifrån var i de tredimensionella världen som objektet befinner sig som beskrevs med x-, y- och z-koordinaterna. Det är dessa variabler som beskriver själva objektets form och hur stort objektet är.

Texnr, som är den sista variabeln som är med i objektstrukturen, är texturbildens nummer. Här beskrivs vilken textur som används till objektet. Det är en integervariabel som skrivs i objektstrukturen och i 3D-motorn kontrollerar programmet vilket texturnummer

```
void toFile(struct objectstruct a){
      FILE *fp;
      fp = fopen(FILENAME, "ab");
      if(fp == NULL){ }
      else{
         fwrite(&a, sizeof(objectstruct), 1, fp);
         fclose(fp);
      }
}
```

Figur 4.2: Skriver objekt till fil.

som objektet har och hämtar rätt bild i en tabell med texturer.

4.3 Skriva och läsa från filen

Objekten som beskrivs med objektstrukten skrivs direkt efter varandra ner till överföringsfilen från bildanalysprogrammet. I figur 4.2 visas koden för funktionen som skriver till filen. För att kunna läsa ut objekten från överföringsfilen så läses block från filen i storleken av en objektstruktur. I figur 4.3 visas koden för funktionen som läser från filen.

3D-motorn använder sig av filen på det sättet att den läser in samtliga objekt in i en objektarray som definierats med ett maximalt antal platser. I 3D-motorprogrammet finns det ett maximalt antal objekt som användas, detta för att inte göra världen som ritas upp för stor.

4.4 Problem med strukturen

Systemet att skriva ner strukter i en fil användes på grund av att de är ett lätt sätt att föra över och läsa in ett objekt åt gången. Vissa problem kan uppstå med detta system om uppbyggnaden av strukturen för lagringen ändras. Om strukturen ändras så medför

```
void fromFile(void)
{
   int x = 0 ;
   FILE *fp;
   fp = fopen(FILENAME, "rb");
   if(fp != NULL){
      while(fread(&worldObject[x], sizeof(objectstruct), 1, fp)){
         x++;
      }
      fclose(fp);
   }
   nrOfObjects = x;
}
```

Figur 4.3: Läser in objekt från fil.

detta att gamla filer som skapats tidigare för visualiseringen av planlösningen i 3D inte kan användas då dessa innehåller den gamla strukturen. Ett sätt att lösa detta är att göra konverteringsprogram som på de tillkomna egenskaperna för objekten lägger in ett standardvärde eller har en omräkning av befintliga värden till uppdaterade värden.

Det finns redan protokoll för att beskriva ett objekt för att använda i 3D-program. Det valdes att inte använda ett redan befintiligt protokoll då det inte skulle vara nödvändligt att kunna ta in annan data till 3D-motorn än den som skapats av bildanalysprogrammet och då detta underlättade för programmeringen av programmen.

5 Bildtolkning

I detta projektet skapades även ett bildanalysprogram. Detta program läser in en bild (planlösning) och ritar upp denna i ett fönster på skärmen. Detta programmet är till för att användaren skall rita upp bilden, till exempel planlösningen, som senare skall sparas ner i en fil och kunna användas för att visualiseras i 3D-motorn som skapades i detta projekt.

Programmet för bildtolkningen används på det sättet att användaren får en muspekare som kan förflyttas över bilden, till exempel planlösningen, och kan med hjälp av denna rita ut objekt. Objekten placeras ut genom att användaren klickar med musen var objektet skall vara placerat och området som objektet skall uppta. Objekten sparas sedan undan i strukturer som representerar objekt, till exempel väggar och fönster.

Detta kapitel har delats in i två avsnitt vilka är musimplementation och bildanalysprogrammet. Det första avsnittet tar upp de åtgärder som fick vidtas när musfunktioner skulle implementeras till programmet för bildtolkning. Avsnittet Bildanalysprogrammet tar upp de olika delar som konstruerades för att kunna tolka bilder och skapa de filer som används av 3D-motorn för att rita upp olika världar.

5.1 Musimplementation

I projektet behöver användaren kunna använda sig av musen i bildanalysprogrammet för att kunna rita upp och markera objekt. För att initiera musen behövs först DirectX's bibliotek DirectInput initieras. I vår funktion initMouse så sker alla standardinställningar för initiering av DirectInput och sedan initiering av musen. För detaljerad information se DirectX SDK dokumentation. Funktionen update används för att uppdatera muspositionen. Den känner av hur mycket musen rört sig i x- och y-led och sen multipliceras dessa värden med ett lämpligt tal som får väljas så att rörelsen på skärmen ser och känns bra för användaren som använder programmet. I funktionen update finns även en så kallad spärr som ser till att musen endast kan röra sig inom vissa uppsatta x- och y-koordinater. Denna

spärr är till för att inte musen skall försvinna för användaren, det vill säga att musen inte får komma utanför skärmen.

Ytterligare en funktion som implementerades i bildanalysgsprogrammet är funktionen drawCursor. Funktionen drawCursor ritar upp muspekaren på skärmen. Det är bilden som användaren ser som muspekaren på skärmen. Muspekaren som används togs fram secifikt för bildanalysprogrammet och ritades med photoshop. Detta för att få muspekaren att fungera bra med de funktionsområden som den används till i bildanalysprogrammet.

Funktionen mouseButtonDown är en boolesk funktion som returnerar true eller false beroende på om den musknapp som frågas efter har tryckts ned eller inte. För att kontrollera höger musknapp skickas värdet 1 (ett) till funktionen, och för att kontrollera den vänstra musknappen skickas värdet 0 (noll).

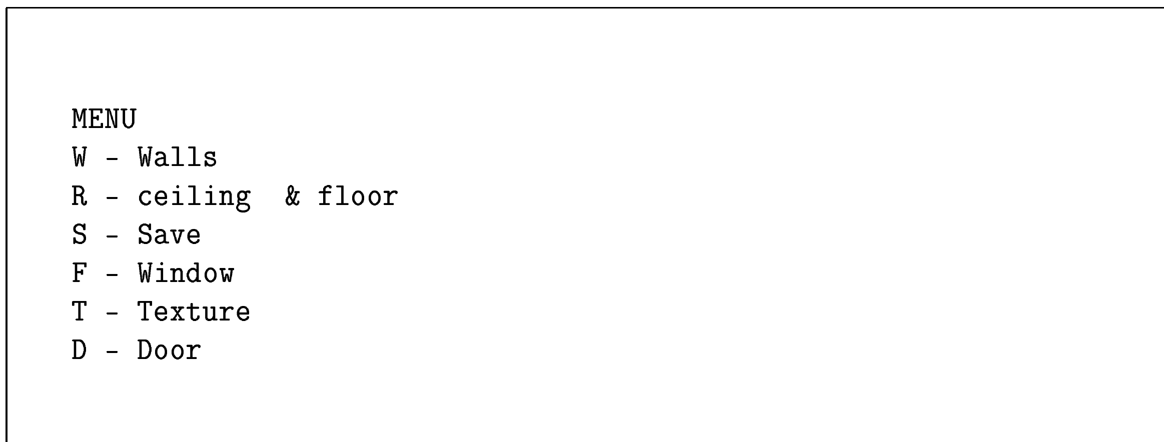
Den sista funktionen som skapades för musimplementation är getMousePos. Funktionen getMousePos ger x- och y-positionerna till två variabler via referens.

5.2 Bildanalysprogrammet

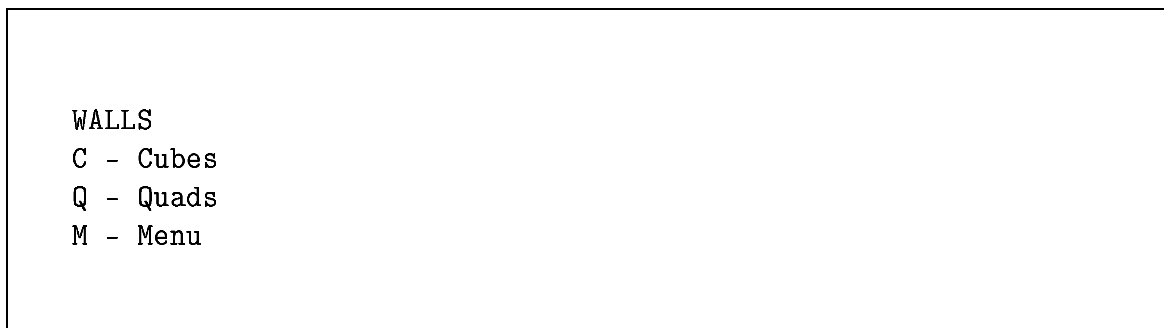
Detta är ett grafiskt program avsedd för att skapa planlösningar som senare konverteras till 3D-objekt som lagras i en fil för att kunna användas till den 3D-motor som konstruerats i detta projektet. Programmet använder en inläst bild på en 2D planlösning som mall för att användaren skall kunna rita/bygga upp 2D världen. Efter sparning kan denna fil användas i tillhörande 3D program för att gå runt i en 3D värld av den planlösning som skapades i 2D.

5.2.1 Meny

När bildanalysprogrammet startas kan en meny ses i översta vänstra hörnet. Denna meny är till för att hjälpa användaren av bildanalysprogrammet att navigera sig runt och använda dess funktioner. Menyvalen görs med hjälp av tangentbordet. Användaren trycker ner den tangent som valet i menyn refererar till. Menyn ser från början ut som visas i figur 5.1.


MENU
W - Walls
R - ceiling & floor
S - Save
F - Window
T - Texture
D - Door

Figur 5.1: Menyns utseende vid programstart.


WALLS
C - Cubes
Q - Quads
M - Menu

Figur 5.2: Menyns utseende för väggfunktioner.

Om användaren trycker ner tangenten 'w' kommer den till en undermeny som behandlar väggar. Där har användaren två alternativ att välja mellan. Menyalternativen för väggarna ser ut enligt figur 5.2.

Alternativet Cubes betyder kuber och alternativet Quads betyder fyrkanter. Här får användaren tänka till lite vad som behöver för typ av objekt. Kuber behövs på de väggar där du har tänkt placera ut fönster eller dörrar. Det är även smart att använda dem på till exempel innerväggar eller dylikt där användaren vill få en upplevelse av tjocklek på väggen. Annars är fyrkanter alternativet att rekommendera då de ritas upp snabbare i 3D motorn. Snabbhet i 3D motorn bör hela tiden eftersträvas för annars kan grafiken *hacka* (ritas upp

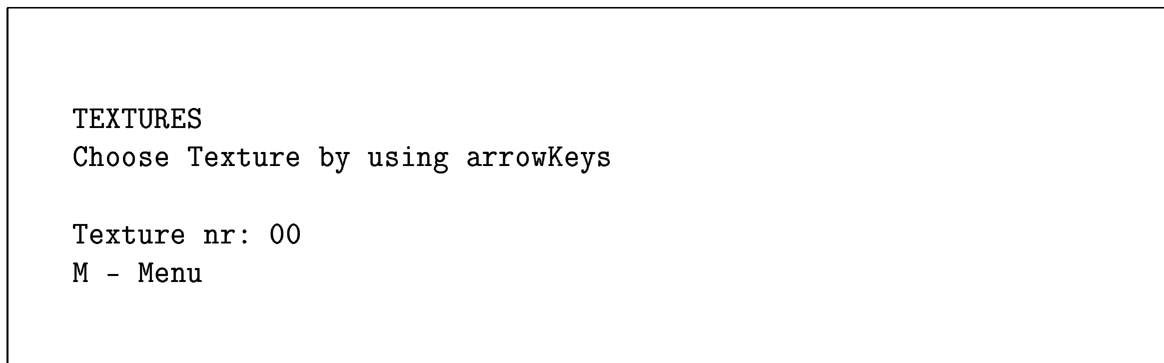
långsamt) och ger då en sämre upplevelse för användaren. I bildanalysprogrammet finns även en bugg[1].

Alternativet ceiling & floor betyder att användaren ges möjlighet att rita ut var tak och golv skall finnas. När användaren ritar ut detta så blir det genomskinlig så användaren ska kunna se allt annat den ritat. Det vill säga att taket blir genomskinligt så användaren fortfarande kan se objekt som skall finnas under taket.

För att spara, sker det med en tangenttryckning av 'S'. Denna knapp bör endast tryckas ned en gång för annars sparas all information ytterligare en gång. Detta system kanske inte är så praktiskt men tyvärr har inte denna funktion utvecklats bättre då den schemalagda projekttiden tog slut.

För att välja fönster trycker användaren ned tangenten 'F'. När användaren ska placera sitt fönster skall användaren placera det inom en tidigare utritad kub. Programmet känner sedan av att det finns ett fönster på den väggen och skapar ett hål och placerar fönstret rätt och med en förbestämd storlek. Så det är egentligen endast den första musklickningen som är den viktiga för placeringen av fönstret då användaren markerar ut var fönstret skall placeras i en vägg. Den andra markeringen användaren gjorde, för placeringen av fönstret, bryr sig programmet inte om. Förbättringar som skulle kunna göras här är att endast låta användaren klicka en gång för att placera ut fönstret. Det finns också en bugg i programmet som kan inträffa när användaren placerar in ett fönster. För att undvika att denna buggen inträffar så skall alla väggar sättas ut först och därefter bör användaren placera ut fönster. Om användaren skulle placera ut en vägg efter att ett fönster har placerats ut så kan det uppstå ett fel. Programmet har ytterligare en begränsning och det är att fönstret har alltid en viss riktning. Detta innebär att fönstret kan se bakvänt ut vid placering i olika väggar. Detta vill säga att om ett fönster placeras så att utsidan för fönstret är åt vänster i skärmen ser fönstret riktigt ut. Däremot om användaren placerar det så att utåt skall vara mot höger i skärmen kommer fönstret att se bakvänt ut.

[1] Denna bugg gör att fyrkanterna ritas upp med den storleken du ritar ut dem med (precis som kuberna), men vid uppritning i 3D motorn så är det endast en fyrkant utan tjocklek som ritas upp.



Figur 5.3: Menyns utseende för texturval.

Trycker användaren ned tangenten 'T' kommer användaren till en undermeny för texturer. Den meny ser ut som visas i figur 5.3. Genom att använda piltangenterna kan användaren bläddra genom tillgängliga texturer. När så görs så ändras numret efter texten Texture nr. Det finns för tillfället 8 st olika texturer att välja mellan i bildanalysprogrammet. Här bör tilläggas att användaren inte kan välja texturer för fönster, dörrar, tak och golv. Dessa texturer är förbestämda. Det är endast för kuber och fyrkanter som olika texturer kan väljas.

Sista snabbkommandot som användaren kan välja är 'D' vilket är till för att placera ut dörrar. Samma princip som vid placering av fönster, det vill säga att de ska placeras inom en utritat kub (vägg). Samma begränsning som vid fönster finns även här, det vill säga att dörrarna kan se bakvända ut vid olika placeringar. Då dörrar är mera symmetriska än fönster medför detta att vid placering av dörrar i vissa riktningar är det inte lika uppenbart med riktningen av objektet.

5.2.2 Uppritning av planlösning och skalning

Det som bör göras innan programmet startar är att byta ut den bild som heter *planlosning.jpg* till den nya bild över den planlösning som önskas behandlas i bildanalysprogrammet. När du startar programmet kommer denna bild att ligga som bakgrund över hela

skärmen.

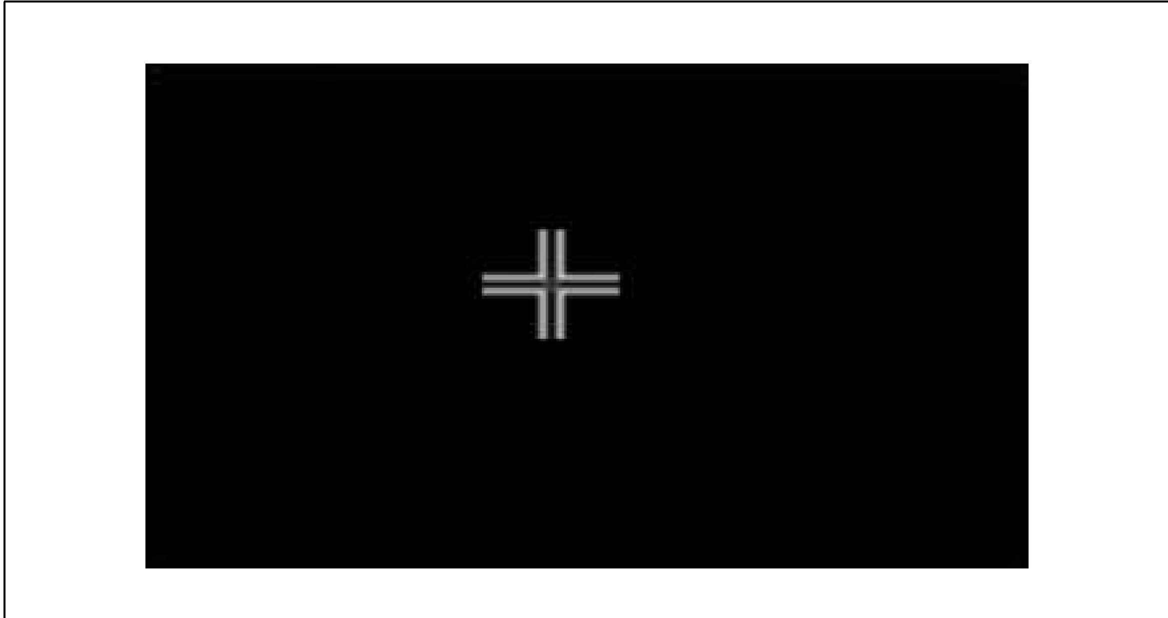
Då en annan planlösning vill användas i programmet krävs det nu att denna bild har samma dimensioner som vår tidigare bild. Detta så att inte bilden blir förvrängd. Tanken i projektet var att alla möjliga storlekar på bilder skulle kunna användas. Det fanns dock inte tid till att konstruera funktioner för att utföra skalningar.

5.2.3 Texturlösning och koordinattolkning av mus

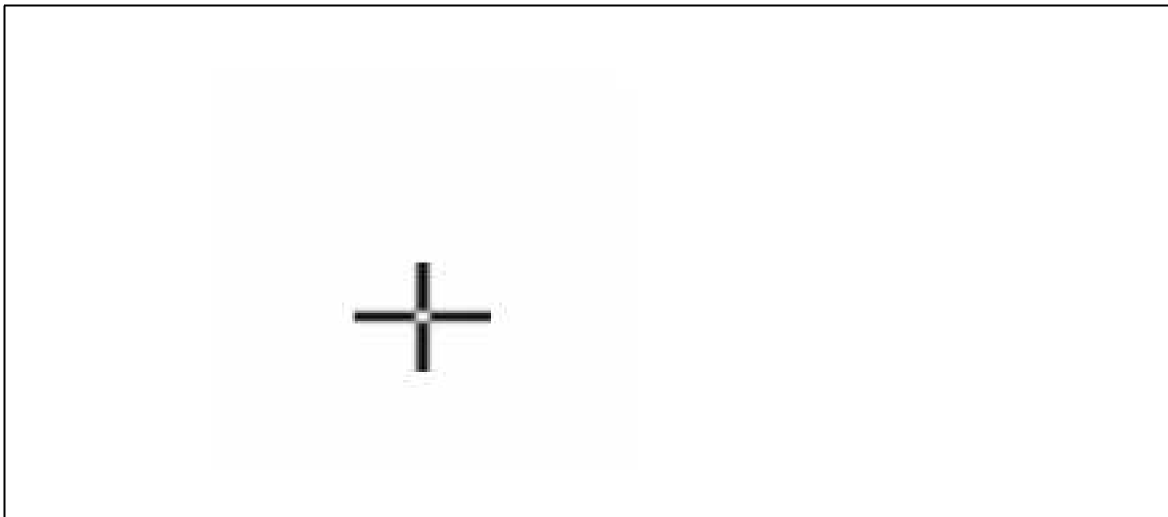
Valet av textur för muspekaren till musen gjordes noga. Det behövdes en muspekare som skulle fungera dels på den vita bakgrunden och dels på svarta eller andra färger på fält. Under utvecklingen av muspekaren testades ett stort antal olika texturer till muspekaren. Det testades både olika färger som mönster för att få en som stämde med det som ville kunna visualiseras med muspekaren. En färg olik mot vit och svart hade löst detta problem, då arbetet med muspekaren sker till större delen över bakgrunder som antingen är svarta eller vita. En svart eller en vit muspekare kändes som det bästa alternativet. Lösningen var att göra pekaren en kombination av ett svart och vitt mönster. När muspekaren är över svart bakgrund så syns bara det vita och när muspekaren är över en vit bakgrund syns det svarta. I figurerna 5.4 och 5.5 visas muspekaren över svart respektive vit bakgrund. I figur 5.6 visas muspekaren över en svart-vit delad yta.

När användaren väljer att rita ut ett objekt i bildanalysprogrammet, till exempel väggar, fönster eller golv, anropas funktionen picProgram. Arrayer för lagring av positioner, bredd, höjd och övriga attribut skickas med som parametrar till funktion picProgram. Objekten fönster, väggar och golv/tak har alla olika arrayer. Funktionen picProgram läser av två stycken av användarens musklickningar. För varje gång en användare använder sig av musen och klickar anropas funktionen getMousePos. Funktionen getMousePos returnerar muspekarens x- och y-position då musen användes. Dessa positioner sparas i arrayerna som skickades med in som parametrar till funktionen picProgram vid funktionsanropet.

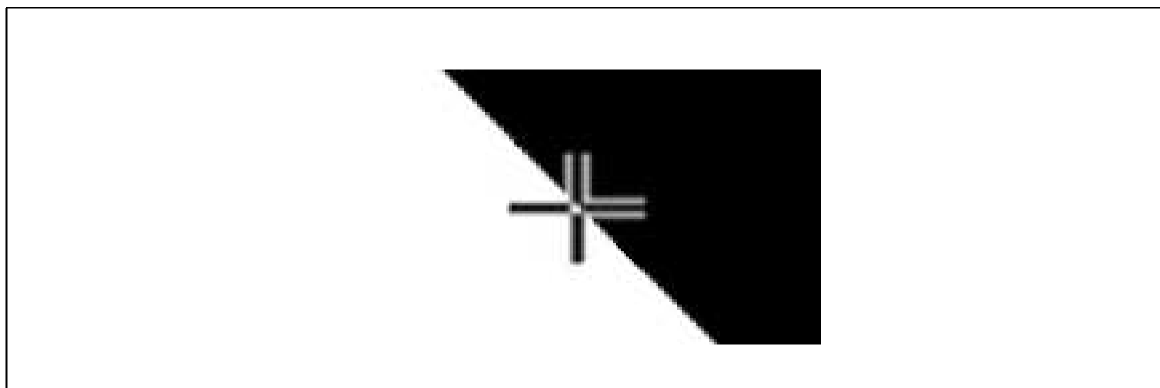
För att rita upp en 2D vägg krävs det att dessa två musklickningar sker på rätt sätt.



Figur 5.4: Över en svart bakgrund ser hårkorset vitt ut.



Figur 5.5: Över en vit bakgrund ser hårkorset svart ut.



Figur 5.6: Över brytningen av en delad svartvit yta.

Den första musklickningen anger övre vänstra startpositionen för fyrkanten. Den andra musklickningen skall vara sådan att den anger fyrkantens nedersta högra hörn. Från dessa två musklickningar räknas sedan höjden och bredden ut. Notera att alla beräkningar och värden är i 2D. Omvandlingen till 3D sker i ett senare skede av programmet. Det sista som sker i denna funktion är ett anrop till funktionen drawQuad som sparar undan värden för den nya fyrkanten i vertexbufferten.

5.2.4 Utritning av 2D objekt på planlösning

Funktionen drawSketch ritar ut alla objekten, det vill säga fönster, väggar och tak, i 2D som har placerats ut. Funktionen loopar igenom alla objekt och sätter tillhörande textur och ritar ut med Direct3Ds funktion DrawPrimitive. Taket ritas ut genomskinligt som tidigare nämndes eftersom det inte skall skymma allt det övriga som ritats ut. Varje typ av objekt har sitt eget index och det är det som avgör hur många varv i loopen som utförs.

5.2.5 Omvandling av 2D till 3D

När menyvalet Save väljs, med tangentbordstryckning på 's', anropas funktionen save(). Denna funktion anropar i sin tur funktionen saveToFile, med olika värden beroende på vilken typ av objekt som skall sparas. I tidigare avsnitt så nämndes att de olika typerna

av objekt som existerade var fönster, dörrar, väggar och tak/golv. Men det finns även indelningar inom dessa objekt. Till exempel delas objektet väggar upp i kuber (cubes) eller fyrkanter (quads). En fyrkant (quad) har inget tjocklek, det vill säga är en tunn skiva. Fyrkanter är ett bra val när väggar skall göras då de ritas upp snabbare än kuber och underlättar för 3D-motorn i dess arbete. Kuber och fyrkanter skapas alltid med en viss förbestämd höjd i 3D världen. Det finns en typ av objekt som kalls SPECIALWALL och den används när ett objekt inte använder denna förbestämda höjd. Här kan användaren bestämma höjden själv. Detta objekt kan användas till alla andra kubiska objekt användaren vill skapa, till exempel fönsterkarmarna eller dörrar och så vidare. När tak sparas så sparas även ett golv tillhörande taket.

Funktionen saveToFile tar in alla parametrar som krävs för att göra omvandlingen till 3D och sen spara undan till fil. Bredd är lika i 2D och 3D. Höjd i 2D blir djup i 3D. Höjd på objektet och placering i höjdled av objekten i 3D kan inte utläsas från 2D-bild utan de värdena hårdkodas som ett standardvärde. Det vill säga att alla väggar startar nere vid golvet så länge användaren inte använt sig av specialwall. Variablerna för x-, y- och z-position kan beräknas fram med enkla beräkningar. Sedan multipliceras alla värden med ett fördefinierat värde för att få rätt skala vid 3D uppritningen.

5.2.6 Multiplicering av textur över objekt

För att en textur skall passa snyggt på ett objekt så får det inte se utdraget ut. Det gäller då att multiplicera ursprungsbilden ett visst antal gånger i höjd- och djup/bredd-led. För att lösa detta på en bra sätt anropas funktionen getTexNr med längden på väggen och ett fördefinierat värde som parametrar. Detta fördefinierade värde beror på texturens storlek och har testats fram för att se till att texturen har kvar sin ursprungliga storlek. Direct3D använder inte pixels för att avgöra bredd/höjd på objekt utan en egen enhet, vilket var orsaken till att ögonmått användes.

Funktionen getTexNr returnerar ett värde som senare används för 'u' eller 'v', där 'u'

är antal multipliceringar i bredd av texturen och 'v' antal multipliceringar i höjd.

5.2.7 Skapa fönster

När användaren valt att skapa ett fönster från menyn får den först sätta ut var fönstret skall vara placerat. När användaren placerar ut fönstret så tar programmet endast hänsyn till den första musklickningen eftersom bredd, höjd och djup är fördefinierade värden som används till alla fönster. Användaren skall tänka på att placera ut fönstret så att det hamnar inom en utritad vägg. Detta för att programmet ska kunna göra plats för fönstret i väggen. Efter detta anropas funktionen createWindow. Det första som sker i createWindow-funktionen är att skapa fönsterkarmar. Sedan söker programmet igenom alla utritade väggar och undersöker om det utritade fönstret ligger inom någon av dessa. Om den finner ett fönster som ligger inuti en vägg så tas denna vägg bort och 4 nya väggar skapas. På så vis skapas ett hål med exakt samma storlek som fönstret, och på samma position som fönstret placerades.

När fönster sparas så definieras det som typen WINDOW. Detta tolkas sedan i 3D motorn och alla objekt med denna typ ritas upp som genomskinliga. I figur 5.7 visas ett fönster i 3D.

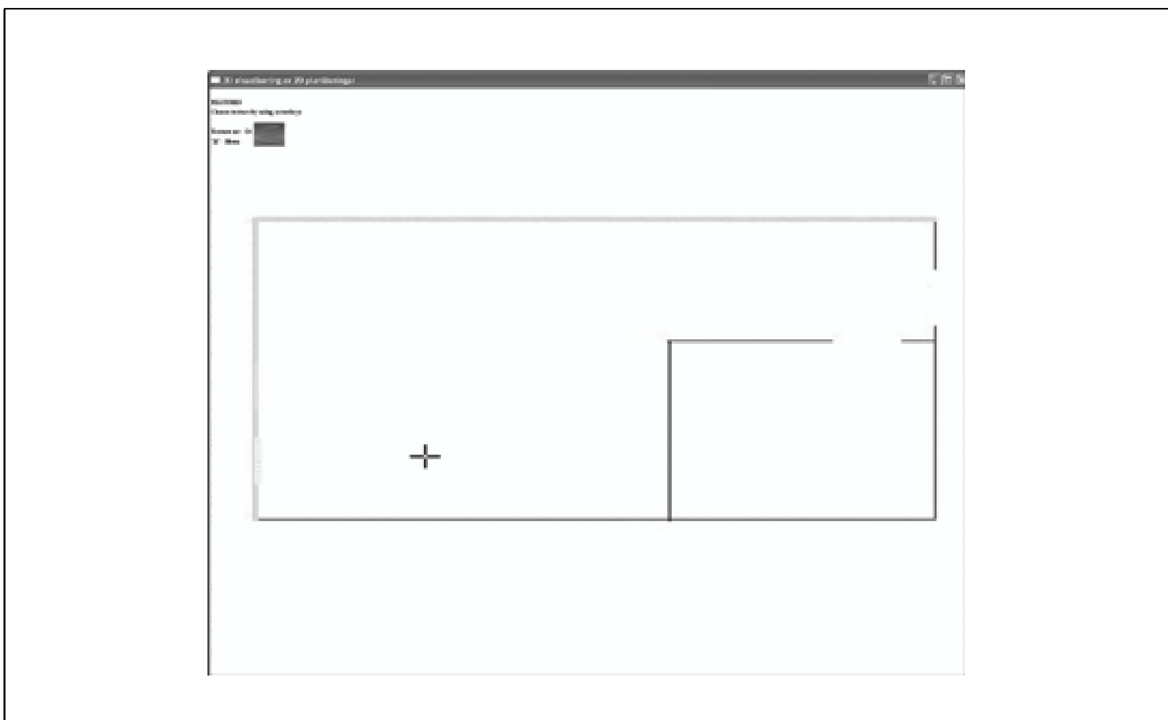
5.3 Vidareutveckling i bildanalysprogrammet

I figur 5.8 visas hur det kan se ut när en användare använder sig av bildanalysprogrammet. Däremot finns ett antal förbättringar som kan göras till bildanalysprogrammet som tas upp i detta avsnitt.

Det finns en bugg som inträffar när fönster placeras ut. Buggen är ganska allvarlig och programmet kan i värsta fall krascha. Men på grund av att den schemalagda tiden för programmeringen tog slut kunde denna bugg ej åtgärdas.

Vid utritning av rektanglar i bildanalysprogrammet så ritas dessa upp på samma sätt som kuberna, dvs att det ser ut som den har en tjocklek (bredd i 2D). Detta är missvisande

Figur 5.7: Ett fönster i 3D-världen.



Figur 5.8: Vy av bildanalysprogrammet.

för i 3D programmet så ritas alla rektanglar upp utan tjocklek.

Fönster och dörrar som sätts ut ser endast korrekta ut i en viss riktning.

Texturer kan ju väljas för väggarna men inte för fönster, dörrar, tak och golv.

Ytterligare en förbättring vore att göra så att användaren kan använda vilken storlek som helst för sin bild på planlösningen. Som det är nu måste bilden vara i en bestämd storlek.

6 Erfarenheter

Projektet har gått bra och de utsatta målen har uppnåtts. Dock har inte extramålen, som sattes ut ifall det skulle finnas mycket tid över på slutet, uppnåtts.

Från början fanns tankar om att planlösningen skulle läsas automatiskt av datorn, det vill säga utan hjälp av användaren. Detta skulle dock bli alltför svårt då det antagligen skulle krävas en del svåra matematiska algoritmer. Istället gjordes programmet så att det läser in planlösningen och sedan får användaren markera ut vart väggarna och andra objekt skall vara placerade utifrån detta sparas koordinaterna till en överföringsfil mellan bildanalysprogrammet och 3D-motorn.

Under den utsatta tiden har arbetet fördelats mycket bra. Lika stor prioritering lades på rapporten som på programmeringen. På så vis så kunde de sista veckorna ägnas åt rättning och finslipning av rapporten. Det medförde även att de programmeringsändringar som gjordes under projektets gång skrevs in till rapporten omgående och inte glömdes bort. Samarbetet i projektet har fungerat bra då projektet har deltats upp mellan rapportskrivning och programmeringsuppgifter så att effektivitet skulle kunna uppnås.

Under projektets tid stöttes det på en hel del problem. Dock inga tillräckligt stora för att de utsatta målen inte nåddes. De flesta problemen var programmeringsproblem som har löst med hjälp av hjälpfiler till DirectX. Alla problem löstes dock inte och de har efterlämnat några buggar i programmet, en av dem som kan få programmet att krascha.

Med detta projekt upptäcktes fördelarna med ett program för att visualisera planlösningar i 3D. Det finns många möjliga användingsområden för ett projekt som detta, både för privatpersoner och företag. Det krävs dock betydlig vidareutveckling av projektet för att det skall kunna användas i kommersiellt syfte.

7 Slutsats

Vid slutförandet av denna rapport är projektet vid den punkten att en användare kan använda sig av både bildanalysprogrammet för att konstruera 3D-världar samt kunna använda de världarna i 3D-motorn. Användaren kan i bildanalysprogramet konstruera två typer av väggar, tak och golv, fönster, dörrar samt välja bland ett antal texturer.

De två typer av väggar som existerar är en vägg där ingen tjocklek på väggen existerar (rektangel), samt en där tjocklek på väggen existerar (kub). Då det i verkligheten inte existerar väggar utan tjocklek kan det anses konstigt att denna möjlighet finns med i bildanalysprogrammet. Rektangelväggar togs med för att förbättra uppritningshastigheten i 3D-motorn. Rektangelväggar är rekommenderat att användas vid ytterväggar då en användare inte kommer att se tjockleken då användaren enbart är inne i en byggnad. Rektanglar kan även användas då en användare vill använda sig av två olika texturer på olika sidor av en vägg mellan två rum. Detta då kubväggarna inte kan ge två texturer på de olika sidorna av väggen. Det kan då placeras två rektangelväggar bredvid varandra och motsvara en kubvägg. Både rektangelväggarna och kubväggarna har en fast höjd. Det finns även en till typ av vägg som är en specialvägg där höjden inte är fast. Specialväggen används då fönster och dörrar placeras ut i en vägg då vägg skall fylla upp omkringliggande område vid fönster och dörrobjekt. Specialväggen går inte att välja med menyval för användaren.

Tak och golv placeras ut samtidigt och går inte att välja textur för dessa. Golv och takobjektet har en fast höjd mellan sig. Denna höjden är den samma som höjden på rektangelväggen och kubväggen.

Fönster och dörrar är två objekt som kan placeras ut i kubväggar. När ett fönster eller en dörr placeras ut i en vägg delas väggen upp och fönstret eller dörren placeras in. Därefter placeras specialväggar över och under fönster eller dörrobjekten. Dörrarna är enbart programmerade så att de kan placeras i väggar som är placerade vertikalt på skärmen. De är även programmerade så att de placeras ut med insidan av objektet åt ett förbestämt håll. Detta medför att fönster kan komma åt fel håll på väggar om de placeras

ut på väggar där ut skall vara åt höger på bildskärmen. Detta gäller även för dörrar.

Texturer kan väljas när väggar skall placera ut. Då väljer användaren texturen innan med texturvalet på menyn i bildanalysprogrammet. Det finns ett antal olika färdiga texturer som kan användas.

Menyvalen i bildanalysprogrammet fungerar på det viset att en användare ser en meny i fönstret i bildanalysprogrammet. De olika alternativen som användaren kan utföra representeras av en bokstav. För att välja trycker användaren ner motsvarande tangent på tangentbordet för alternativet i menyn användaren vill utföra. Det finns en grundmeny samt ett antal olika undermenyer. Ett exempel på en undermeny är den som användaren kommer till efter väggar har valts i grundmenyn. I denna undermeny kan användaren välja vilken typ av vägg som skall användas (rektanglar eller kuber).

I menyn kan användaren spara den värld som ritats upp till en fil. Till filen skrivs då objekten beskrivna av objektstrukturer.

3D-motorn är konstruerad för att visualisera de objekt som kan ritas ut med bildanalysprogrammet. Från filen, som kan sparas i bildanalysprogrammet, läser 3D-motorn in objekten och lagrar dessa i en buffer. Det är från denna buffer som 3D-motorn sedan hämtar objekt och ritar upp i fönstret för 3D-motorn.

3D-motorn är väldigt beroende av hur kraftfull dator som används. Då 3D-motorn konstruerades gjordes detta vid viss en dator och senare vid tester med andra datorer har det visats att 3D-motorn fungerar olika bra på olika datorer.

Referenser

[1] Kevin Hawkins and Dave Astle, *OpenGL game programming*, Rocklin, Calif. : Prima Tech; London : Pearson Education, 2001. ISBN 0-7615-3330-3

[2] Mason McCuskey, *Special Effects Game Programming with DirectX*, Premier Press, 2002. ISBN 1-931841-06-3.

[3] http://www.microsoft.com, *Microsoft Homepage*, 5 maj 2004.

[4] http://www.gamedev.net/reference/programming/features/d3do/page4.asp, *An Overview of Direct3D*, 5 maj 2004.

A Bilaga A - Funktionsbeskrivning

A.1 Funktioner från 3D-motorn

A.1.1 drawScene

Syntax: void drawScene()

Pre: Skall ha anropat fromFile() och initWorld().

Post: 3D-världen har blivit uppritad.

Parametrar: -

Beskrivning:

A.1.2 WinMain

Syntax: int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int CmdShow)

Pre: true

Post: Programmet avslutas och tar bort pekare.

Parametrar: *hInstance* - ett slags ID så att Windows kan skilja detta program från andra.
hPrevInstance - den här kan ignoreras. Det är en parameter som betydde något i 16-bits windows men inte längre.
lpcmdLine - det här är en ny form av det gamla argc, argv[] variablerna i det gamla main. lpcmdLine är en sträng som innehåller allting som användaren skriver i commando-prompten förutom namnet på vårt program.
nCmdShow - Beskriver hur fönstret skall startas upp. T.ex. om det ska startas minimerat eller maximerat.

Beskrivning: Kör igång 3D-motorn

A.1.3 direct3dInit

Syntax: bool direct3DInit(CWindow *pwindow)

Pre: windows skall vara initierat.

Post: direct3D har blivit initierat.

Parametrar: **pwindow* - pekare till vår windows klass

Beskrivning: initierar Direct3D

A.1.4 setupView

Syntax: void setupView()

Pre: direct3D ska vara initierat

Post: inställningar för vy, fokuseringspunkt och hur långt som skall renderas i 3D-världen har blivit satta.

Parametrar: -

Beskrivning: Sätter inställningar för vy , fokuseringspunkt samt hur långt som skall renderas i 3D-världen.

A.1.5 setupTexture

Syntax: void setupTexture()

Pre: direct3D ska vara initierat

Post: inställningar för texturer har blivit inställda

Parametrar: -

Beskrivning: sätter inställningar för texturer

A.1.6 release

Syntax: void release()

Pre: true

Post: har frigjort pekarna dxDev och dxD3D

Parametrar: -

Beskrivning: frigör pekarna dxDev och dxD3D

A.1.7 releaseVB

Syntax: void releaseVB()

Pre: true

Post: har frigjort pekare till vertexbuffers

Parametrar: -

Beskrivning: frigör pekare till vertexbuffer

A.1.8 setupRenderState

Syntax: void setupRenderState()

Pre: direct3D ska vara initierat

Post: har satt inställningar för uppritningen.

Parametrar: -

Beskrivning: sätter inställningar för uppritningen

A.1.9 setMaterial

Syntax: void setMaterial()

Pre: direct3D ska vara initierat

Post: egenskaper för material har satts

Parametrar: -

Beskrivning: egenskaper för material sätts

A.1.10 setLight

Syntax: void setLight()

Pre: direct3D ska vara initierat

Post: har satt inställningar för ljussättningen.

Parametrar: -

Beskrivning: sätter inställningar för ljussättningen.

A.1.11 create

Syntax: void create(float _x, float _y, float _z, float _nx,float _ny,float _nz, float _u, float _v)

Pre: true

Post: har tilldelat värden till variablerna i klassen CVertex.

Parametrar: *Vbnr (vertex buffer nummer)* - ett nummer i en pekararray. Dessa pekare är pekare till en så kallad vertex buffers som DirectX använder för att lagra information om objektet som skall ritas.

u, v - Detta är egenskaper för texturen på objektet. En textur är ytan på objektet, det vill säga att olika bilder kan användas för att simulera mönster och utseende på objektet. Texturen kan väljas att läggas en gång över objektet eller alternativt multipla gånger. Det medför att om en textur läggs en gång över objektet dras texturen ut för att passa objektet. Detta kan i vissa fall medföra att texturen ser utdragen och oproportionerlig. Då kan istället upprepningar av texturen användas för att täcka objektet så att texturen inte dras

ut. u beskriver antal upprepningar i x-led av texturen. v beskriver antal upprepningar i y-led av texturen.
x, y, z - Vilken position objektets översta vänstra hörn har i 3D-världen.
width, height, deep - bredd, höjd och djup på objektet. Notera att parametern deep sätts till noll vid anrop till funktionen som beskriver fyrkanterna.
Beskrivning: tilldelar värden till variablerna i klassen CVertex.

A.1.12 createWindow

Syntax: void createWindow(HINSTANCE hInstance, int CmdShow)
Pre: ska anropas från en WinMain funktion
Post: har skapat ett fönster
Parametrar: *hInstance* - denna parameter kommer ursprungligen från Winmain och det är ett slags ID för ditt program.
CmdShow - denna parameter kommer ursprungligen från Winmain. Beskriver hur fönsstret skall startas upp. Tex om det ska startas minimerat eller maximerat.
Beskrivning: skapar ett fönster

A.1.13 wndProc

Syntax: static LRESULT CALLBACK wndProc(HWND wpHWndm, UINT msg, WPARAM wParam, LPARAM lParam)
Pre: windows skall vara initierat
Post: funktionen gör ingenting för oss men den kan ta hand om meddelanden från Windows.
Parametrar: *wpHWndm* - variabeln hwnd från windows. För att identifiera om man har olika fönster.

Msg - ID på meddelandet

wParam, lParam - det här är i meddelandet ligger.

Beskrivning: funktionen gör ingenting för oss men den kan ta hand om meddelanden från Windows.

A.1.14 GetHwnd

Syntax: HWND GetHwnd()

Pre: Direct3D skall vara initierat

Post: har returnerat hwnd

Parametrar: -

Beskrivning: returnerar hwnd

A.1.15 drawCube

Syntax: void drawCube(int vbnr, float _u, float _v, float _x, float _y, float _z, float width, float height, float deep)

Pre: true

Post: har definierat punkter för angivet objekt i en vertexbuffer.

Parametrar: *Vbnr (vertex buffer nummer)* - ett nummer i en pekararray. Dessa pekare är pekare till en så kallad vertex buffers som DirectX använder för att lagra information om objektet som skall ritas.

u, v - Detta är egenskaper för texturen på objektet. En textur är ytan på objektet, det vill säga att olika bilder kan användas för att simulera mönster och utseende på objektet. Texturen kan väljas att läggas en gång över objektet eller alternativt multipla gånger. Det medför att om texturen läggs en gång över objektet dras texturen ut för att passa objektet. Detta kan i vissa fall medföra att texturen ser utdragen och oproportionerlig. Då

kan istället upprepningar av texturen användas för att täcka objektet så att texturen inte dras ut. u beskriver antal upprepningar i x-led av texturen. v beskriver antal upprepningar i y-led av texturen.
x, y, z - Vilken position objektets översta vänstra hörn har i 3D-världen.
width, height, deep - bredd, höjd och djup på objektet. **Beskrivning:**

A.1.16 drawQuad

Syntax: void drawQuad(int vbnr, float _u, float _v, float _x, float _y, float _z, float width, float height)
Pre: true
Post: har definierat punkter för angiven fyrkant i en vertexbuffer.
Parametrar: *Vbnr (vertex buffer nummer)* - är ett nummer i en pekararray. Dessa pekare är pekare till en så kallad vertex buffers som DirectX använder för att lagra information om objektet som skall ritas.
u, v - Detta är egenskaper för texturen på objektet. En textur är ytan på objektet, det vill säga att olika bilder kan användas för att simulera mönster och utseende på objektet. Texturen kan väljas att läggas en gång över objektet eller alternativt multipla gånger. Det medför att om texturen läggs en gång över objektet dras texturen ut för att passa objektet. Detta kan i vissa fall medföra att texturen ser utdragen och oproportionerlig. Då kan istället upprepningar av texturen användas för att täcka objektet så att texturen inte dras ut. u beskriver antal upprepningar i x-led av texturen. v beskriver antal upprepningar i y-led av texturen.
x, y, z - Vilken position objektets översta vänstra hörn har i 3D-världen.
width, height - bredd och höjd på fyrkanten.
Beskrivning: definierar punkter för angiven fyrkant och sparar dem i en vertexbuffer

A.1.17 drawWorld

Syntax: void drawWorld()

Pre: initWorld skall ha anropats innan

Post: 3D världen är uppritad

Parametrar: -

Beskrivningar: ritar upp 3D världen

A.1.18 fromFile

Syntax: void fromFile(void)

Pre: måste finnas en fil skapad.

Post: har läst in data från fil och sparat i struktar.

Parametrar: -

Beskrivningar: läser in data från fil och sparar i struktar.

A.2 Funktioner från bildanalysprogrammet

A.2.1 saveTofile

Syntax: void saveToFile(int vbnr, int texnr, float u, float v, float x, float y, float z, float width, float height, float deep, float height2Start, float height2, int type)

Pre: true

Post: har konverterat värden till 3D och sen sparat till fil.

Parametrar:

Vbnr - Ett index på vertexbuffer.

u,v - beskriver hur många gånger texturen skall multipliceras på ytan.

x,y,z - positioner i 2D. z värdet är lite överflödigt, sätt detta till noll vid anropet.

width, height, deep - bredd, höjd i 2D. deep är ett överflödigt värde eftersom 2D behandlas. Sätt denna till noll vid anropet.
height2ToStart - Detta anger y position i 3D.
height2 - Detta anger höjd i 3D.
type - Typ på objektet. Antingen WALL = 1 eller WINDOW = 2.

A.2.2 picProgram

Syntax: void picProgram(float xpositions[], float ypositions[], float width[], float height[], int &index, int &index2, int type, int texture[], int vbnr[], float height2Start[], float height2[])
Pre: true
Post: Användaren har klickat ut två positioner med musen och dessa har sparats I arrayer.
Parametrar:
xpositions, ypositions - I dessa arrayer sparas positionerna.
width, height - I dessa arrayer sparas bredd och höjd på objekten användaren ritat ut på skärmen.
index, index2 - Två st index för att hålla reda på vart i arrayerna man befinner sig. Index hör till positionsarrayerna och index2 hör till bredd och höjd arrayerna.
type - Om det skall vara CUBE=1 eller QUAD=11 eller SPECIALWALL=2;

A.2.3 drawSketch

Syntax: void drawSketch()
Pre: picProgram måste ha anropats först och användaren ha ritat ut nåt.
Post: Objekt som användaren har klickat ut har ritats upp.
Parametrar:

A.2.4 menu

Syntax: void menu()

Pre: true

Post: en meny har ritats upp i programmet.

Parametrar:

A.2.5 initFont

Syntax: void initFont

Pre: true

Post: användning av grafisk text i programmet har initierats.

Parametrar:

A.2.6 getTexNr

Syntax: float getTexNr(float nr, float value)// **Pre:** true

Post: ett värde har returnerats som kan användas som 'u' eller 'v'.

Parametrar:

nr - längden på väggen.

value - ett värde som du skickar beroende på storleken på texturen. Om det är en 64*64 bild så skicka med 3.0 här för retur till 'u' och 1.69 för retur till 'v'.

A.2.7 defineWallExactly

Syntax: void defineWallExactly(float xpos, float ypos, float xpos2, float ypos2, float width, float height, int type, int texture, float height2Start, float height2)

Pre: true

Post: har sparat en vägg i en vertex buffer.

Parametrar:

xpos1, ypos1 - x- och y position för översta vänstra hörnet på väggen I 2D.

xpos2, ypos2 - x- och y position för nedersta högra hörnet på väggen I 2D.

width, height - bredd och höjd på objektet i 2D.

type - Om det skall vara CUBE=1 eller QUAD=11 eller SPECIALWALL=2;

texture - nummer på den textur som ska vara på väggen.

height2ToStart - Detta anger y position i 3D.

height2 - Detta anger höjd i 3D.

A.2.8 createDoor

Syntax: void createDoor()

Pre: picProgram skall ha anropats först.

Post: Om dörren befinner sig inom en utplacerad vägg så görs plats i väggen för dörren. Annars sker ingenting.

Parametrar:

A.2.9 splitWall2

Syntax: void splitWall2(int &i , int &kk)

Pre: createdoor skall ha anropats först.

Post: Om dörren befinner sig inom en utplacerad vägg så görs plats i väggen för dörren. Annars sker ingenting.

Parametrar:

i - ett index som håller reda på var användaren befinner sig i positionsarrayerna.

kk - ett index som håller reda på var användaren befinner sig i bredd- och höjdarrayerna.

A.2.10 createWindow

Syntax: void createWindow()

Pre: picProgram skall ha anropats först.

Post: Om fönstret befinner sig inom en utplacerad vägg så görs plats i väggen för fönstret. Annars sker ingenting.

Parametarar:

A.2.11 splitWall

Syntax: void splitWall(int &i , int &kk)

Pre: createWindow skall ha anropats först.

Post: Om fönstret befinner sig inom en utplacerad vägg så görs plats i väggen för fönstret. Annars sker ingenting.

Parametrar:

i - ett index som håller reda på var användaren befinner sig i positionsarrayerna.

kk - ett index som håller reda på var användaren befinner sig i bredd- och höjdarrayerna.

A.2.12 save

Syntax: void save()

Pre: true. picProgram för dock ha anropats först. Annars finns inget att spara.

Post: alla objekt som användaren har ritat ut har blivit sparade till fil.

A.3 Musfunktioner

A.3.1 initMouse

Syntax: void initMouse(HWND hwnd, HINSTANCE hinst)

Pre: true

Post: directInput och musen har initierats.

Parametrar:

hwnd, hinst - är båda parametrar som skapades när windows initierades.

A.3.2 update

Syntax: void update()

Pre: initMouse skall ha anropats först.

Post: uppdaterar muspekarens position.

Parametrar:

A.3.3 drawCursor

Syntax: void drawCursor(DirectX *directx, int vbnr, int texnr)

Pre: initMouse skall ha anropats först.

Post: Muspekaren har blivit uppritad.

Parametrar:

Directx - en pekare till directx klassen.

vbnr - vertex buffer nummer.

Texnr - nummer på den textur som valts som muspekare.

A.3.4 mouseButtondown

Syntax: bool mouseButtonDown(int Button)

Pre: initMouse skall ha anropats först.

Post: returnerar true eller false beroende på om den efterfrågade musknappen har tryckts ned eller inte.

Parametrar:

Button - ett heltalsvärde som avser vilken knapp som tryckts ned. 0 = vänster musknapp, 1 = höger musknapp.

A.3.5 getMousePos

Syntax: void getMousePos(float &xpos,float &ypos)

Pre: initMouse skall ha anropats först.

Post: x- och y-position har skickats som referens.

Parametrar:

xpos,ypos - två variabler där x- och y-positionerna sparas.

Beskrivning: skillnaden mellan getMousePos och getMousePos3 är hårfin men den första används i picProgram för den första musklickningen och getMousePos3 används för den andra musklickningen. Båda funktionerna är anpassade till dessa respektive klickningar och ger dem optimal precision.

A.3.6 getMousePos3

Syntax: void getMousePos(float &xpos,float &ypos)

Pre: initMouse skall ha anropats först.

Post: x- och y-position har skickats som referens.

Parametrar:

xpos,ypos - två variabler där x- och y-positionerna sparas.

Beskrivning: skillnaden mellan getMousePos och getMousePos3 är hårfin men den första används i picProgram för den första musklickningen och getMousePos3 används för den andra musklickningen. Båda funktionerna är anpassade till dessa respektive klickningar och ger dem optimal precision.