Lab 1

Getting started with Open GL

Kurs: Datorgrafik 2016

Labbrapport inlämnad: 8 November 2016

Gruppnummer:

1

Gruppmedlemmar:

Victor Svensson
920101-2714
Martin Hjalmarsson
860313-8218

Levererad till:

Daniel Canelhas

Uppgift 1.1 – Hello Gpu

Använd glGetString funktionen för att få namnet på rendreraren och vilken version av Open GL som stöds.

Utskriften från glGetString ger följande:

Renderer	AMD Radeon R9 200 Series
Open GL version supported	4.4.12874
Compability Profile Context	14.100.0.0

Hitta hårdvaruspecifikationen för rendreraren

AMD Radeon R9 200 Series säger bara vilken serie grafikkortet tillhör. Med hjälp av vendor ID och model ID funnet i enhetshanteraren kunde information om den specifika modellen hämtas.

Graphic card model	R9 290X
Video RAM	4GB
Number of processing units	2816
GPU Clock frequency	1000 mhz
Memory bandwith	352 / GB/s

Vilka funktioner saknas från den här versionen av Open GL jämfört med den senaste versionen?

Den nya Open GL versionen 4.5 har något som kallas ”Direct State Access”(DSA). Den medför att man kan modifiera OpenGL objekt utan att behöva binda dom till kontexten. I praktiken innebär det att istället för att kalla t.ex. **glGenVertexArrays**, **glBindVertexArray** osv så kan du kalla **glCreateVertexArrays** direkt. Det gör att man kan arbeta mer objektorienterat.

Uppgift 1.2 – Your first triangle

Beskriv funktionerna som används i programmet.

glGenVertexArrays	Genererar ett namn för vertex arrayen som skickas in
glBindVertexArray	Sätter given vertex array till den aktiva vertex arrayen
glGenBuffers	Genererar ett namn för buffern som skickas in
glBindBuffer	Sätter given buffer till den aktiva buffern

glBufferData	Laddar upp data till den aktiva buffern
glEnableVertexAttribArray	Aktiverar det givna attributet
glVertexAttribPointer	Specificerar ett attribut
glCreateShader	Skapar en shader av given typ
glShaderSource	Länkar den givna shadern till en shader-kod(sträng)
glCompileShader	Kompilerar shader-koden
glCreateProgram	Skapar ett tomt program som shaders kan bindas till
glAttachShader	Binder given shader med givet program
glLinkProgram	Länkar ihop programmet med bundna shaders. I gruppens fall: Skapar executables för vertex shader och för fragment shader.
glDeleteShader	Flaggar en shader för radering. Är shadern ej binden till något program så raderas den direkt. Annars så flaggas den för senare radering.
glUseProgram	Programmet installeras och läggs I current render state

Vilka dimensioner I screen-space mappas dina X,Y och Z koordinater till?

X	Positiv åt höger på skärmen
Y	Positiv uppåt på skärmen
Z	Positivt utåt från skärmen

Vad är gränserna för Z och vad händer om du går utanför den gränsen?

Gränserna är -1 och +1. Är du utanför detta intervall så clippas bilden.

Uppgift 1.3 – Introduction to shaders

Varför får fragment shadern olika värden på varje punkt I triangeln? Kan man kontrollera detta beteende?

Den får olika värden för varje fragment på grund ut av interpolation. Man kan kontrollera det här beteendet genom att använda olika interpolation qualifiers. Default är smooth. Ändrar man till flat så får hela triangeln färgen av den provokerande vertexen (den sista av de tre som bildar triangeln).

Uppgift 1.4 – Passing parameters to shader programs

Inget att redovisa under detta avsnitt.

Uppgift 1.5 – 3D geometry

Varför används index array?

Man sparar utrymme. Storleken på 60 shorts (index tabell) och 12 floats (vertex) är totalt 168 byte. Storleken på 60 floats, det vill säga om man skickar in alla vertex som används inklusive repetitioner, är 240 byte.

Uppgift 1.6 – Model, view & projection transformations

Varför är multiplikationsordningen viktig?

Multiplikation av matriser är inte kommutativ, dvs A*B != B*A.

Vad händer om man multiplicerar matriserna I omvänd ordning?

Om man utför multiplikation i omvänd ordning kommer resultatet inte att blir det förväntade. Om man tex utför perspektivmultiplikationen först så har man komprimerat z till värden mellan -1 och 1 vilket kommer att göra att resterande transformationer kommer ge ett konstigt resultat.

Uppgift 1.7 – Linear algebra isn’t fun

Hittills har glDepthFunc varit sätt till GL_LESS. Vilka andra alternativ finns det för denna parameter?

Parametern används för att styra hur djupsorteringen fungerar. Ordet “godkänd” betyder I detta sammanhang att pixeln ritas ut och ett nytt Z-värde för den pixeln blir sparat I Z buffern.

GL_NEVER	Godkänns aldrig
GL_LESS	Godkänns om pixelns Z värde är mindre än det sparade Z värdet.
GL_EQUAL	Godkänns om pixelns Z värde är samma som det sparade Z värdet
GL_LEQUAL	Godkänns om pixelns Z värde är mindre eller lika som det sparade Z värdet
GL_GREATER	Godkänns om pixelns Z värde är större än det sparade Z värdet
GL_NOTEQUAL	Godkänns om pixelns Z värde inte är samma som det sparade värdet
GL_GEQUAL	Godkänns om pixelns Z värde är större eller lika som det sparade Z värdet
GL_ALWAYS	Godkänns alltid

Är det originala värdet att föredra? Varför? Vad kan hända annars?

Eftersom att man vill rita ut det som ligger närmare kameran överst så vill man att objekt med ett mindre z-värde ska ritas ut över objekt med ett högre z-värde. Väljer man t.ex. GL_GREATER så kan det bli så att små bakgrundsobjekt ritas över stora förgrundsobjekt och det ser konstigt ut.