01418382 สภาพแวดล้อมการทำงานคอมพิวเตอร์กราฟิกส์

Computer Graphics Working Environment

การบรรยายครั้งที่ 12

การเชื่อมต่อ OpenGL เข้ากับโปรแกรม Cg และ ตัวอย่างโปรแกรม Cg

ภาคต้น ปีการศึกษา 2554

ผู้สอน: อ.ชาคริต วัชโรภาส

เกริ่นนำภาษา Cg

ภาษา Cg

- Cg ย่อมาจาก C for Graphics
- Cg ถูกพัฒนาขึ้นโดยบริษัท Nvidia ด้วยความร่วมมือกับบริษัท Microsoft
- Cg เป็นภาษาโปรแกรมที่มีโครงสร้างภาษาคล้ายกับภาษา C และ C++
- Cg ใช้สำหรับเขียนโปรแกรมเพื่อคำนวณแสงเงาในงานเรนเดอร์ (*high-level shading language*)
- Cg ใช้เขียนได้ทั้ง vertex program และ fragment program ที่จะรันบน *Programmable Vertex Processor* และ *Programmable Fragment Processor* ตามลำดับ ซึ่งหน่วยประมวลผลทั้ง 2 นี้อยู่บน GPU
- Cg มีลักษณะคล้ายกับภาษา HLSL ที่ถูกพัฒนาโดย Microsoft ซึ่งถูกออกแบบมาไว้เพื่องาน shading เช่นกัน

ภาษา Cg (ต่อ)

- แม้ว่า Cg จะมีลักษณะคล้ายภาษา C แต่ Cg ก็มีลักษณะที่แตกต่างจาก C อาทิเช่น ชนิดข้อมูลที่มีมากชนิดขึ้นไป รวมถึงลักษณะการทำงานของโปรแกรม Cg ที่มุ่งเน้นเพื่องานคำนวณด้านกราฟิกส์ (ซึ่งต่างจากภาษาระดับสูงอื่นซึ่งถูกใช้เพื่องานในลักษณะทั่วไป)
- Cg ถูกออกแบบเพื่อให้ผู้ใช้สามารถใช้ความสามารถของอุปกรณ์กราฟิกส์ได้อย่างเต็มที่ โดยยังสามารถใช้ภาษาระดับสูงในการควบคุมการทำงาน
- เมื่อโปรแกรม Cg ถูกคอมไพล์แล้วนำไปรันบน GPU แล้วนั้น โปรแกรมจะประมวลผลข้อมูลที่เข้ามาใน pipeline ด้วย thread ซึ่งมีจำนวนมากและทำงานในแบบขนาน
- แต่ละ thread จะทำงานด้วยโค้ดจากโปรแกรมเดียวกัน แต่ประมวลผลจากข้อมูลเข้าใน thread แต่ละตัวด้วยข้อมูลที่แตกต่างกัน

ภาษา Cg (ต่อ)

- ข้อมูลที่ถูกเข้าไปในโปรแกรมเป็นข้อมูลที่ถูกส่งเข้ามาใน pipeline
- ข้อมูลถูกส่งผ่านจาก CPU เข้ามาใน GPU
- ข้อมูลเหล่านี้สามารถอยู่ในรูปของ vertex data และ pixel data (ดูสไลด์การบรรยายครั้งที่ 3 เพื่อทบทวน)
- แต่ในวิชานี้ เราจะประมวลผลข้อมูลที่เป็น vertex data เท่านั้น เพื่อใช้ในการคำนวณการสร้างภาพ 3 มิติ (3D Rendering)
- ส่วนการประมวลผลข้อมูลที่อยู่ในรูป pixel data (ซึ่งอยู่นอกขอบเขตของรายวิชานี้) จะใช้เพื่อการประมวลผลบนรูปภาพได้ (Image Processing)

การเชื่อมต่อ OpenGL เข้ากับโปรแกรม Cg

การเชื่อมต่อ OpenGL เข้ากับโปรแกรม Cg

- ในการใช้งาน OpenGL เพื่อสร้างภาพให้ปรากฏบนหน้าจอ ข้อมูลที่เป็น vertex data ของวัตถุจะถูกส่งเข้าไปใน pipeline และผ่านการประมวลผลตามขั้นตอนใน pipeline ให้ได้ผลการคำนวณที่จะถูกเก็บอยู่ใน framebuffer
- ถึงแม้ว่าขั้นตอนการคำนวณใน OpenGL pipeline (โดยเฉพาะที่ใช้ในการคำนวณแสงเงา) จะมีลักษณะเป็นแบบ fixed-function อุปกรณ์กราฟิกส์ถูกออกแบบโดยยอมให้ผู้ใช้เขียนโปรแกรมควบคุมการทำงานภายใน GPU ได้
- ภาษา Cg ถูกออกแบบให้สามารถทำงานร่วมกับ OpenGL และ DirectX ได้อย่างสะดวก ซึ่งง่ายต่อผู้ใช้ในการใช้งาน
- เราจะมาเรียนรู้เพียงวิธีการเชื่อมต่อ OpenGL เข้ากับโปรแกรม Cg โดยนำโปรแกรม OpenGL ที่เคยเขียนไว้แล้วมาปรับเพื่อใช้งานร่วมกับ Cg

การเชื่อมต่อ OpenGL เข้ากับโปรแกรม Cg (ต่อ)

- อุปกรณ์กราฟิกส์ในปัจจุบัน (ตั้งแต่ ค.ศ. 2002 เป็นต้นมา) จะถูกออกแบบให้ภายใน GPU ประกอบด้วยทั้งหน่วยประมวลผลที่เป็นแบบ
  - Non-programmable hardware units
  - Programmable hardware units
- Programmable hardware units ซึ่งเป็นส่วนที่ผู้ใช้สามารถเขียนโปรแกรมเข้าไปควบคุมการทำงานภายในได้ มีส่วนที่เป็น *Programmable Vertex Processor* และ *Programmable Fragment Processor*
- เราเรียกโปรแกรมที่ทำงานอยู่บน *Vertex Processor* ว่า “vertex program” และบน *Fragment Processor* ว่า “fragment program”
- ใน OpenGL มี vertex program และ fragment program ที่ถูกเขียนขึ้นไว้ ซึ่งจะถูกโหลดเข้าไปในหน่วยประมวลผลทั้ง 2 ณ ตอนที่ OpenGL ถูกใช้งาน

การเชื่อมต่อ OpenGL เข้ากับโปรแกรม Cg (ต่อ)

- หากผู้ใช้ต้องการให้โปรแกรมของตนเข้าไปทำงานแทนของ OpenGL ก็สามารถทำได้เช่นกัน
- เนื่องจากอุปกรณ์กราฟิกส์ถูกพัฒนาอย่างต่อเนื่อง อุปกรณ์กราฟิกส์แต่ละรุ่นของแต่ละบริษัทมีโอกาสที่ขีดความสามารถในการทำงานจะไม่เท่ากัน
- ภาษา Cg จึงมีการกำหนดลักษณะความสามารถของอุปกรณ์ หรือที่เรียกว่า Hardware Profile เพื่อให้ผู้ใช้ระบุขีดความสามารถของอุปกรณ์ที่โปรแกรมของตนต้องการใช้งาน
- นอกจากนี้ ผู้ใช้สามารถตรวจสอบ Hardware Profile ของอุปกรณ์เพื่อปรับการทำงานของโปรแกรมตนให้รองรับกับอุปกรณ์ในอดีต หรือในบางกรณีให้รองรับกับอุปกรณ์ในอนาคตได้

ลำดับขั้นตอนการเชื่อมต่อ

- การเขียนโปรแกรมหลัก (ซึ่งในที่นี้เป็นโปรแกรมที่ทำงานบน CPU) เพื่อส่งโปรแกรม Cg ไปทำงานอยู่บน GPU นั้น ผู้ใช้ต้องทำการ
  - คอมไพล์โปรแกรม Cg ให้รองรับการทำงานของ Hardware Profile ที่ต้องการ โดยให้อยู่ในรูปของ object code ซึ่งจะถูกโหลดเข้าไปทำงานภายใน GPU ต่อไป
  - เชื่อมต่อโปรแกรมหลักกับโปรแกรม Cg เพื่อให้โปรแกรมหลักสามารถปรับเปลี่ยนค่าพารามิเตอร์ที่โปรแกรม Cg จะนำไปใช้ในการประมวลผลข้อมูลใน pipeline ต่อไป
- Cg Runtime เป็นไลบรารีที่ช่วยให้ขั้นตอนที่กล่าวมาข้างต้นสามารถทำได้ในขณะที่โปรแกรมหลักกำลังทำงานอยู่ (at runtime)
- การที่ทำขั้นตอนดังกล่าว ณ runtime มีข้อดีในลักษณะที่ผู้ใช้ไม่ต้องคอมไพล์โปรแกรม Cg สำหรับ profile ต่างๆ ไว้ล่วงหน้า รวมถึงไม่จำเป็นต้องจัดเก็บ object code ของโปรแกรมที่ถูกคอมไพล์ล่วงหน้าเหล่านี้ด้วย

การใช้งาน Cg Runtime

การใช้งาน Cg Runtime

- โปรแกรมหลักต้อง include ไฟล์ดังต่อไปนี้

```
#include <Cg/cg.h>
#include <Cg/cgGL.h>
```

- หลังจากนั้น โปรแกรมหลักต้องสร้าง context ซึ่งเป็นโครงสร้างข้อมูล (data structure) ที่ใช้จัดเก็บข้อมูลของโปรแกรม Cg รวมทั้งโปรแกรมที่ถูกคอมไพล์และค่าตัวแปรพารามิเตอร์ โดยคำสั่งที่เรียกใช้ฟังก์ชันมีลักษณะดังนี้

```
CGcontext cgCont = cgCreateContext();
```

- ขั้นตอนต่อไปเป็นการเลือกใช้ profile ซึ่งเราจะเลือกใช้วิธีที่ตรวจสอบ profile ที่ดีที่สุดของอุปกรณ์กราฟิกส์ที่มีอยู่ในระบบ โดยการเรียกใช้ฟังก์ชัน

```
CGprofile vProf = cgGLGetLatestProfile(CG_GL_VERTEX);
```

- ค่าพารามิเตอร์ที่ส่งผ่านฟังก์ชันเป็นการระบุคลาสของ profile ที่ใช้ตรวจสอบ อาทิเช่น `CG_GL_VERTEX`, `CG_GL_FRAGMENT` และ `CG_GL_GEOMETRY`

การใช้งาน Cg Runtime (ต่อ)

- การคอมไพล์โปรแกรม Cg สามารถทำได้หลายลักษณะ ดังตัวอย่างเช่น

```
CGcontext cgCont = cgCreateContext();
CGprofile vProf = cgGLGetLatestProfile(CG_GL_VERTEX);
CGprogram vProg = cgCreateProgramFromFile(cgCont,
  CG_SOURCE,"vertex.cg",vProf,"main",NULL);
```

- ตัวอย่างนี้เป็นคอมไพล์โปรแกรมที่มีลักษณะดังนี้
  - `cgCont` เป็นตัวแปรที่เก็บ `context`
  - `CG_SOURCE` บอกให้รู้ว่า source code ถูกเก็บอยู่ในไฟล์
  - ไฟล์ `vertex.cg` เป็นไฟล์ที่บรรจุ source code
  - คอมไพล์โปรแกรมโดยใช้ profile ดังที่ถูกเก็บไว้ในค่าตัวแปร vProf
  - ชื่อฟังก์ชันเริ่มต้นการทำงาน (entry function) ของโปรแกรม Cg ที่อยู่ในไฟล์
  - สตริงที่เก็บค่าตัวเลือกที่ใช้ในการคอมไพล์ ซึ่งในตัวอย่างนี้ไม่ได้มีการระบุตัวเลือก

การใช้งาน Cg Runtime (ต่อ)

- หลังจากโปรแกรม Cg ได้ถูกคอมไพล์แล้ว โปรแกรมหลักต้องโหลด object code ที่คอมไพล์ได้ผ่านเข้าไปยัง OpenGL โดยใช้ฟังก์ชัน

```
cgGLLoadProgram(vProg);
```

- ก่อนที่โปรแกรม Cg จะถูกเรียกใช้งานในการประมวลผลข้อมูลที่จะผ่านเข้าไปใน pipeline ในรอบการส่งถัดไป ผู้ใช้จำเป็นต้องเรียกใช้ฟังก์ชันเพื่อผูกโปรแกรม Cg ที่เขียนขึ้นนี้ให้เข้าไปทำงานแทนโปรแกรมเดิมของ OpenGL ที่ถูกใช้ในการประมวลผลข้อมูล โดยผู้ใช้ต้องเรียกใช้ฟังก์ชัน

```
cgGLBindProgram(vProg);
```

- นอกจากนี้ ผู้ใช้ต้องทำให้ profile ที่ต้องการใช้งานถูกกำหนดสถานะให้เริ่มใช้งานได้ โดยใช้คำสั่ง

```
cgGLEnableProfile(vProf);
```

- ขั้นตอนทั้งหมดที่กล่าวมาแล้วจะทำให้โปรแกรม Cg ได้เริ่มประมวลผลข้อมูลที่ถูกส่งเข้ามาใน pipeline

การส่งผ่านค่าพารามิเตอร์

- โปรแกรมหลักสามารถส่งผ่านค่าพารามิเตอร์ไปยังโปรแกรม Cg โดยผ่านชื่อของพารามิเตอร์ในโปรแกรม Cg
- ขั้นตอนแรก handle ของพารามิเตอร์ที่จะใช้ในการติดต่อกับตัวแปรพารามิเตอร์ในโปรแกรม Cg ต้องถูกหาค่าจากการเรียกใช้ฟังก์ชัน

```
Cgparameter vLen = cgGetNamedParameter(vProg,"len");
```

- กำหนดค่าพารามิเตอร์ทำได้โดยผ่าน `cgSetParameter{1234}{ifd}v()` ซึ่งสามารถถูกเรียกใช้ในลักษณะต่างๆ อาทิเช่น

```
cgSetParameter1f(vLen,10.5);
```

การตรวจสอบข้อผิดพลาดในโปรแกรม Cg

- โปรแกรมหลักสามารถตรวจสอบข้อผิดพลาดที่เกิดขึ้นในโปรแกรม Cg หรือระหว่างขั้นตอนการเรียกใช้งาน Cg runtime โดยเรียกตรวจสอบสถานะล่าสุดของการทำงานภายในของ Cg ว่ามีข้อผิดพลาดหรือไม่

```
CGerror cgErr = cgGetError();
```

- หากมีข้อผิดพลาดเกิดขึ้น ค่า `cgErr` จะมีค่าไม่เท่ากับ `CG_NO_ERROR` ซึ่งเป็นค่าคงที่ที่ถูกกำหนดไว้

- เราสามารถตรวจสอบข้อผิดพลาดโดยดูจากข้อความ ด้วยการเรียกดูจาก

```
const char *str = cgGetLastErrorString(&cgErr);
```

แบบฝึกหัด

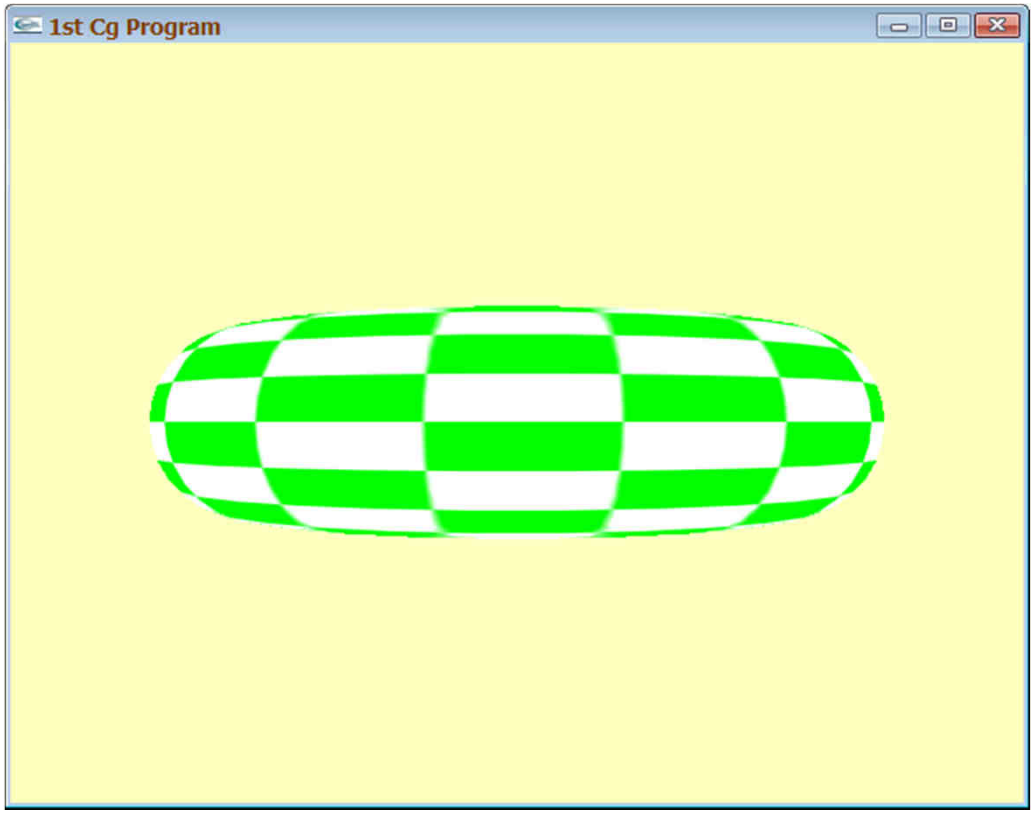
- ให้ดาวน์โหลดไฟล์ SimpleCg.zip จากเว็บไซต์ของวิชา
- ทำความเข้าใจในโค้ด โดยเฉพาะในส่วนของการควบคุมโปรแกรมผ่านคีย์บอร์ดและการส่งผ่านค่าพารามิเตอร์ไปยังโปรแกรม Cg
- คอมไพล์และรันโปรแกรมเพื่อดูการแสดงผลของโปรแกรม
- หากมีปัญหาในการคอมไพล์หรือมีข้อผิดพลาดเกิดขึ้นจากการรัน ให้ลองแก้ปัญหา โดยดูจากภาคผนวก A และ B ของหนังสือ The Cg Tutorial ซึ่งดูออนไลน์ได้ที่ http://developer.nvidia.com/node/76 หากยังแก้ปัญหาไม่ได้ ให้ส่งภาพหน้าจอที่มีปัญหาผ่านอีเมลมายังผู้สอน

คำอธิบายเกี่ยวกับโปรแกรมใน SimpleCg.zip

รายละเอียดเกี่ยวกับโปรแกรม

- ไฟล์ SimpleCg.zip ประกอบด้วยไฟล์ที่สำคัญดังนี้
  - opengl_with_cg.c เป็นไฟล์ของโปรแกรมหลัก
  - cg_programs.cg เป็นไฟล์ที่มี vertex program และ fragment program อยู่
  - donut.tri เป็นไฟล์โมเดลที่เป็นรูปโดนัท (torus shape)
  - Cg_Program.dev เป็นไฟล์โปรเจคของ Dev-Cpp
- หน้าที่หลักในการทำงาน
  - สร้างวินโดว์ขนาด 640 x 480 พิกเซล พร้อมทั้งกำหนด callbacks ในการใช้งาน
  - สร้าง 2D texture ที่เป็นตารางหมากรุกสีขาวสลับเขียวให้พร้อมใช้งานในโปรแกรม
  - โหลดโมเดลจากไฟล์ donut.tri แล้วเก็บไว้ใน Display List หมายเลข 10
  - คอมไพล์ vertex program และ fragment program ที่อยู่ในไฟล์ cg_programs.cg เตรียมไว้เพื่อให้พร้อมใช้งาน ซึ่ง profile ที่กำหนดไว้นั้นยังถูก disable อยู่

ผลการรันในระดับเบื้องต้น



- ในการทำงานช่วงแรกของโปรแกรม OpenGL ถูกใช้ในการแสดงผลที่ปรากฏดังรูป โดยที่ทั้ง vertex program และ fragment program ที่ใช้ประมวลผลข้อมูลใน pipeline ยังคงเป็นโปรแกรมของ OpenGL เอง

ผลการรันในระดับเบื้องต้น (ต่อ)

- โปรแกรมได้ถูกออกแบบให้ vertex program และ fragment program ที่ผู้ใช้ได้เขียนไว้ในไฟล์ cg_programs.cg ถูกนำมาใช้ทำงานแทนของ OpenGL โดยผ่านการกดคีย์ ‘c’ เพื่อให้ profile ที่เตรียมไว้ได้ถูก enable เกิดขึ้น ซึ่งผลการทำงานที่เกิดจากโปรแกรม Cg ที่เขียนไว้นั้นถูกแสดงดังที่ปรากฏในรูป

โปรแกรม Cg ที่อยู่ในไฟล์ cg_programs.cg

- ทั้ง vertex program และ fragment program ที่อยู่ใน cg_programs.cg ถูกเขียนขึ้นเพื่อแสดงให้เห็นถึงรูปแบบตัวอย่างของโปรแกรม Cg โดยยังไม่ได้มุ่งเน้นที่ความถูกต้องในการทำงานตามหน้าที่ใน graphics pipeline

```
void vertex_prog (in float4 i_pos : POSITION,
                  in float4 i_col : COLOR,
                  out float4 o_pos : POSITION,
                  out float4 o_col : COLOR,
                  uniform float param)
{
   o_pos = i_pos;
   o_col = i_col;
}

void fragment_prog (in float4 i_col : COLOR,
                    out float4 o_col : COLOR,
                    uniform sampler2D texture_map)
{
   o_col = i_col;
}
```

โปรแกรม Cg ที่อยู่ในไฟล์ cg_programs.cg (ต่อ)

- ถ้าเราพยายามทำความเข้าใจจากโปรแกรม Cg ทั้ง 2 โปรแกรม ในระดับเบื้องต้น เราพอสรุปการทำงานได้ว่า
  - ตัวอย่าง vertex program จะทำงานโดยเพียงส่งผ่านข้อมูลที่เป็น ***ตำแหน่ง*** และ ***สี*** ของ vertex ที่ไหลเข้ามาจาก pipeline และส่งข้อมูลนั้น โดยไม่ได้มีการเปลี่ยนแปลงออกต่อไปยังส่วนต่อไปใน pipeline
  - ส่วน fragment program ก็ทำงานในลักษณะเดียวกันที่รับข้อมูล ***สี*** ของ fragment ที่เข้ามาจาก pipeline และก็ส่งข้อมูลนั้นออกไปยัง pipeline
- หากสังเกต จะเห็นได้ว่า vertex program มีตัวแปรพารามิเตอร์ *param* ที่มีประเภทข้อมูลเป็น float ซึ่งเป็นตัวแปรที่ถูกกำหนดขึ้นเพื่อใช้รับค่าที่โปรแกรมหลักจะส่งค่าผ่านเข้ามายังโปรแกรม Cg
- ส่วน fragment program ก็มีตัวแปรพารามิเตอร์ *texture_map* ที่มีประเภทข้อมูลชนิดพิเศษที่เป็น sampler2D ซึ่งเป็นตัวแปรที่ใช้อ้างอิงถึง texture ที่ถูกโหลดไว้โดยโปรแกรมหลัก ซึ่ง fragment program ก็สามารถเข้าถึงได้ด้วย

โปรแกรม Cg ที่อยู่ในไฟล์ cg_programs.cg (ต่อ)

- ในการบรรยายครั้งต่อไป เราจะมาดูโครงสร้างภาษา Cg ในส่วนที่เพิ่มจากภาษา C โดยทั่วไป รวมถึงฟังก์ชันภายใน Cg ที่ผู้ใช้สามารถเรียกใช้งานได้
- ให้นิสิตอ่านมาล่วงหน้าในส่วนของบทที่ 2 เรื่อง The Simplest Programs และบทที่ 3 เรื่อง Parameters, Textures, and Expressions ในหนังสือ The Cg Tutorial