01418382 สภาพแวดล้อมการทำงานคอมพิวเตอร์กราฟิกส์
การบรรยายครั้งที่ 1

ประมุข ขันเงิน
pramook@gmail.com

วิชานี้สอน...

- ให้คุณสามารถเขียนโปรแกรมทางคอมพิวเตอร์กราฟิกส์สามมิติของตัวเองได้
  - ใช้ภาษา C++
  - ควบคุมการ์ดจอด้วย OpenGL และ GLSL
- ให้คุณมีความรู้ทางเทคนิคพื้นฐานเกี่ยวกับคอมพิวเตอร์กราฟิกส์
- เราไม่ได้สอนเกี่ยวกับการสร้างงานศิลปะหรือสเปเชียลเอฟเฟกต์

ผู้สอน

- ประมุข ขันเงิน
  - อีเมล์: pramook@gmail.com, fscipmk@ku.ac.th
  - โทรศัพท์: 08-5453-5857
  - ออฟฟิศ: ห้องไม่มีเบอร์เยื้องสำนักงานภาค
  - เวลาเข้าพบ: พุธและศุกร์ เวลา 13.00 น. ถึง 16.00 น. หรือนัดหมายล่วงหน้า

การให้คะแนน

- การบ้าน 40%
  - เขียนโปรแกรม มีประมาณ 4 ครั้ง
- สอบกลางภาค 30%
- สอบปลายภาค 30%
- เกณฑ์การให้คะแนนอ่านเปลี่ยนแปลงได้ในอนาคต
- ตัดคะแนนอิงกลุ่ม
- ตัดคะแนนรวมหมู่ 1 และ 200 (หมู่ 200 มีคนเรียนน้อย)

ความรู้พื้นฐาน

- รู้ภาษา C++
- เขียนโปรแกรมได้ดี
  - วิชานี้มีเขียนโปรแกรมเยอะ
- ความรู้พีชคณิตเชิงเส้น
  - เวกเตอร์ในปริภูมิสามมิติ
  - พหุนาม
  - มีทวนให้

หนังสือ

- David Shreiner et al. **OpenGL Programming Guide: The Official Guide to Learning OpenGL.**
  - ดาวน์โหลดได้ที่ http://fly.cc.fer.hr/~unreal/theredbook/
- เวบไซต์สอน OpenGL และ GLSL ต่างๆ
  - http://www.lighthouse3d.com/opengl/index.shtml
  - http://nehe.gamedev.net/

เวบเพจ

- http://access.cs.sci.ku.ac.th/~pramook/382
- เช็คบ่อยๆ
- การบ้านจะให้ไว้ในเวบเพจนี้เท่านั้น
- ไม่พิมพ์มาให้

นโยบาย

- การบ้านทุกการบ้าน **คุณต้องทำเอง**
  - เขียนโปรแกรมต้องพิมพ์เอง
  - การบ้านข้อเขียนต้องเขียนเอง ด้วยลายมือของตัวเอง
- **ห้ามลอก**
  - ถ้าลอกจะไม่ได้คะแนนสำหรับการบ้านนั้น **ทั้งคนลอกและคนให้ลอก**
  - ห้ามลอกโปรแกรมจากในอินเตอร์เน็ตหรือหนังสือด้วย
- ถามเพื่อนได้ อ่านจากอินเตอร์เน็ตหรือหนังสือได้
  - บอกด้วยว่าทำงานกับใคร
  - บอกแหล่งอ้างอิงด้วย

คอมพิวเตอร์กราฟิกส์

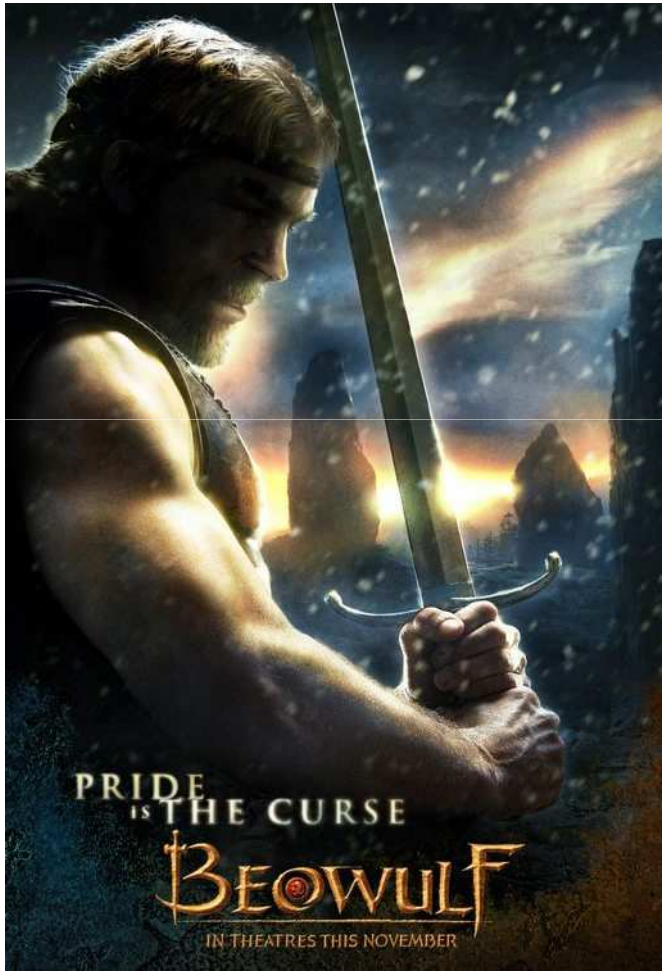
คอมพิวเตอร์กราฟิกส์

- การใช้คอมพิวเตอร์เพื่อสร้างและจัดการสื่อวิทัศน์
- ประโยชน์
  - ความบันเทิง: ภาพยนตร์, เกมส์
  - การศึกษา: ซิมูเลชัน, สื่อประสม
  - อุตสาหกรรม: CAD/CAM

ภาพยนตร์


JOHNNY DEPP
ORLANDO BLOOM
KEIRA KNIGHTLEY
PIRATES of the CARIBBEAN
DEAD MAN'S CHEST

ภาพยนตร์

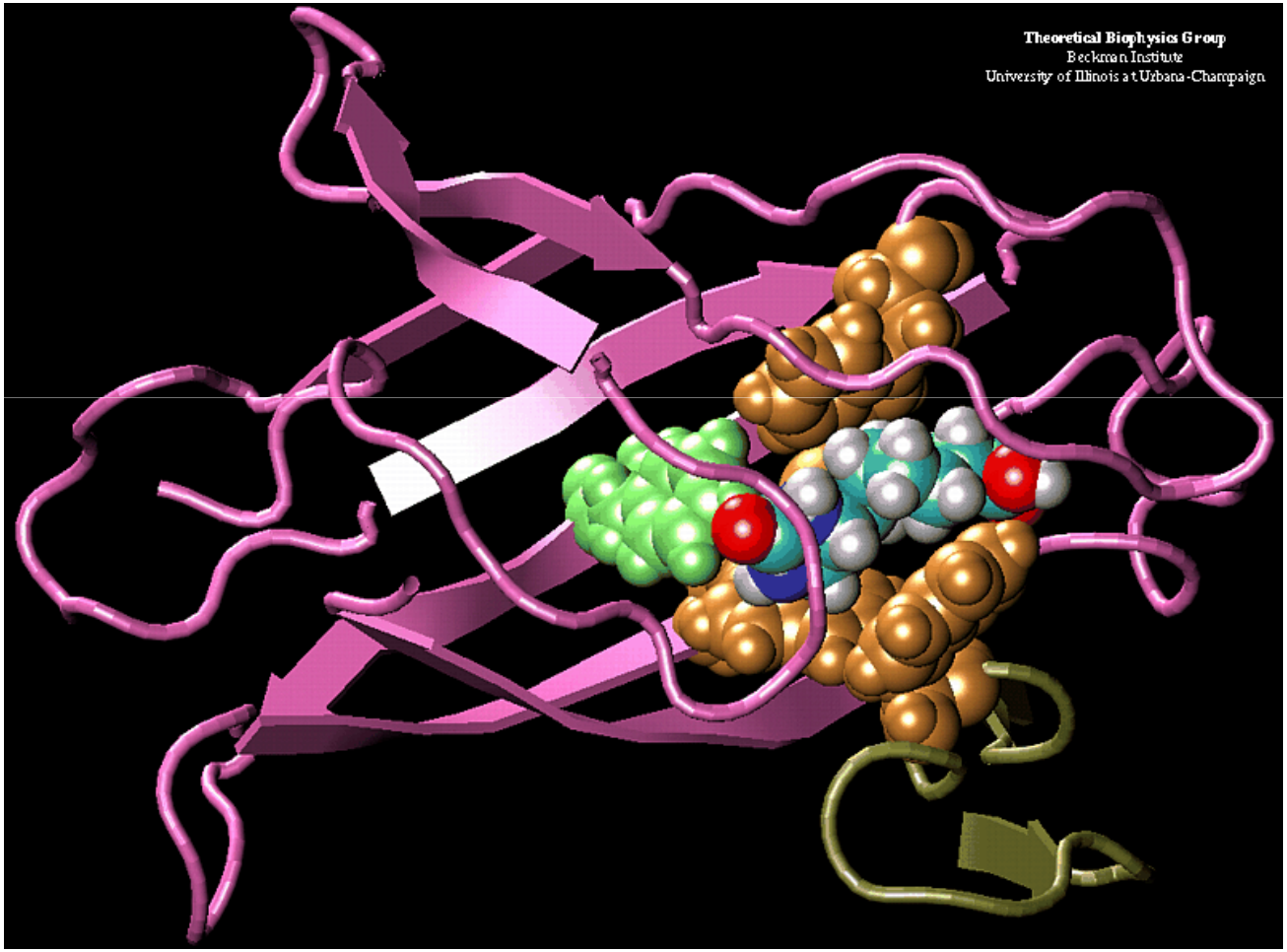


ภาพยนตร์

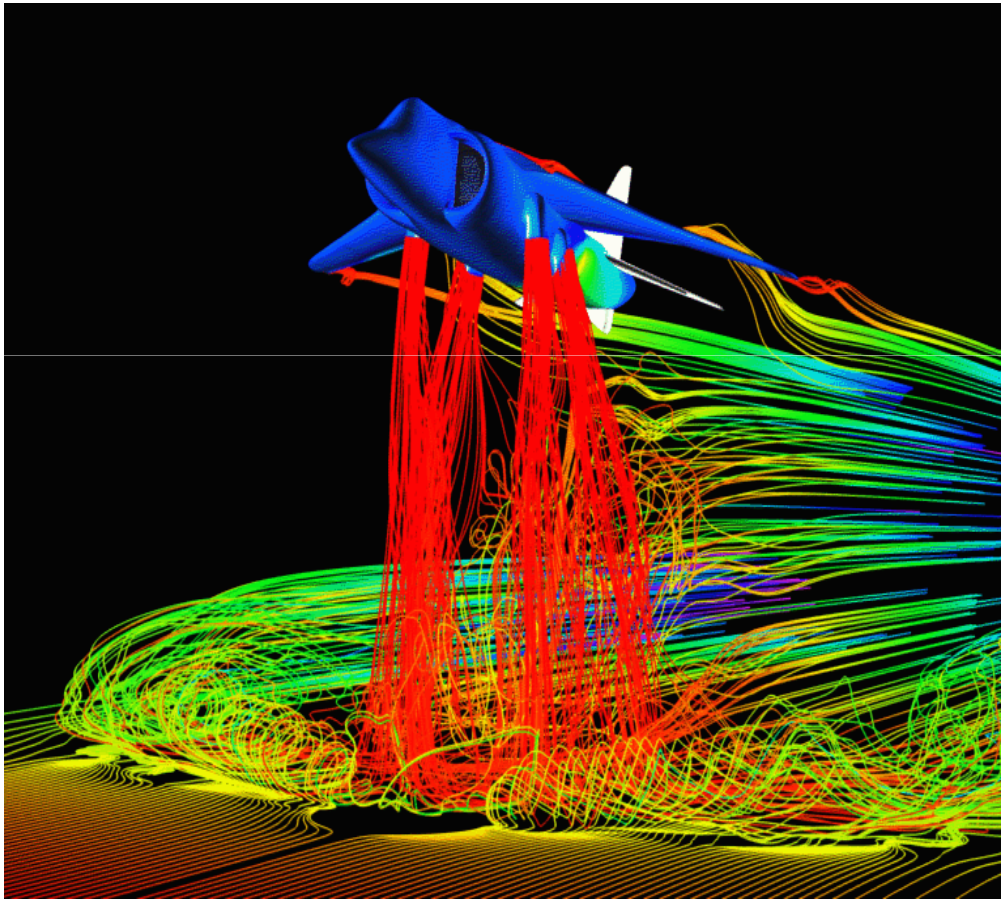
Monster Inc.

Final Fantasy: The Spirit Within

การแสดงภาพทางวิทยาศาสตร์

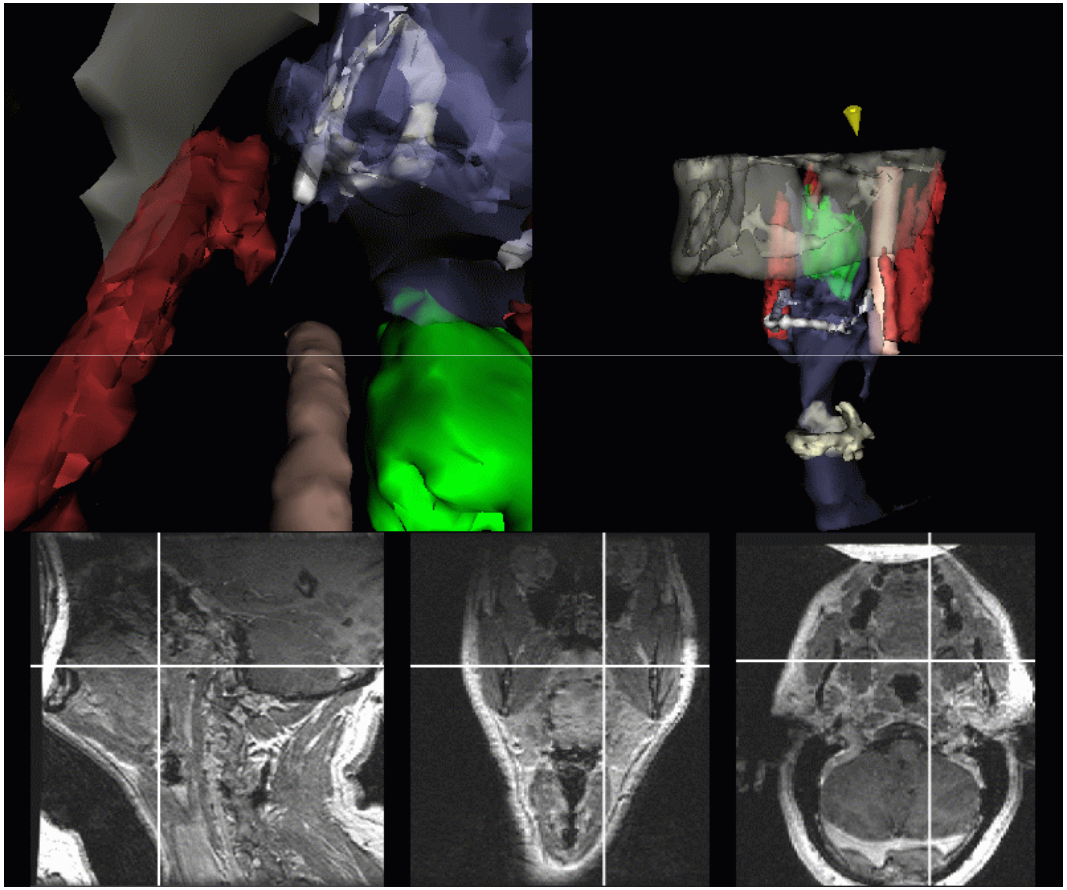


การแสดงภาพทางวิทยาศาสตร์ (ต่อ)

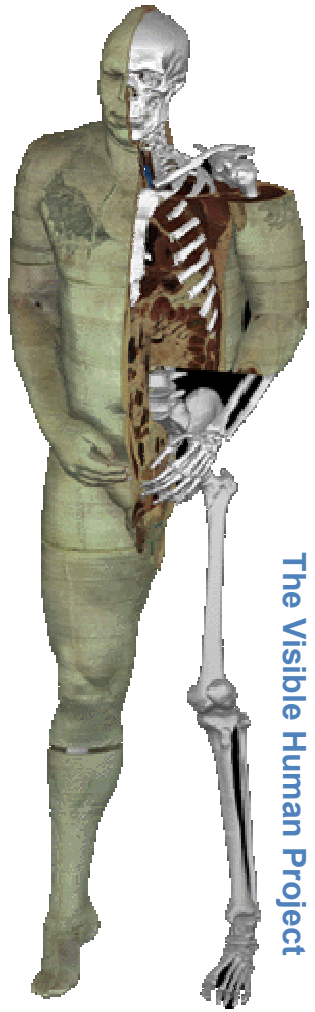


Airflow around a Harrier Jet *(NASA Ames)*

การแสดงภาพทางการแพทย์

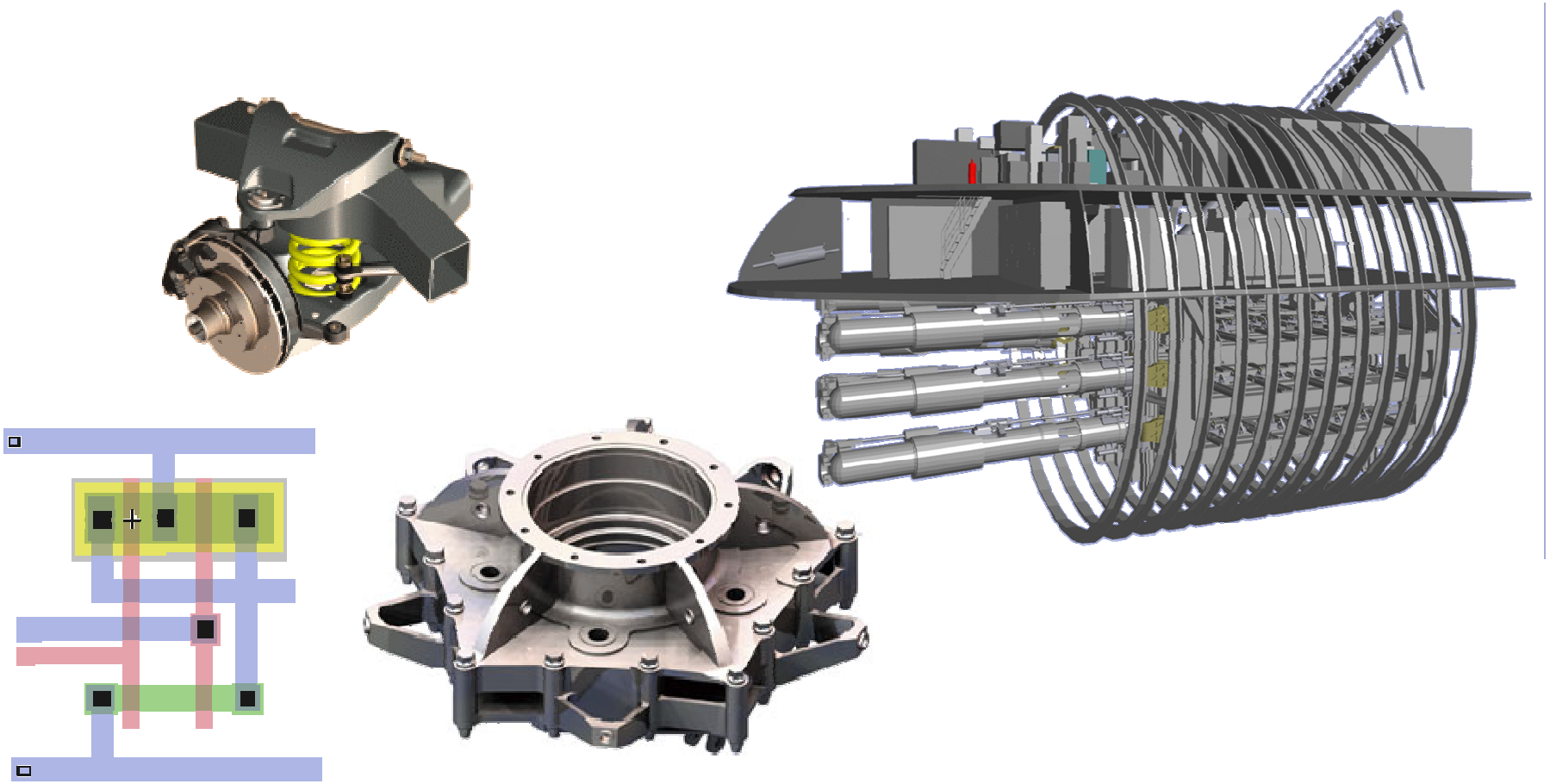


MIT: Image-Guided Surgery Project



The Visible Human Project

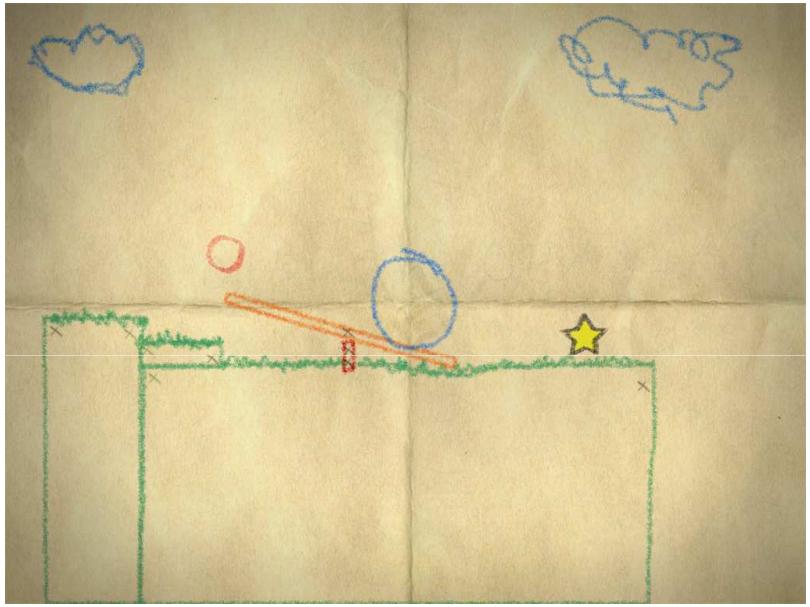
Computer Aided Design (CAD)

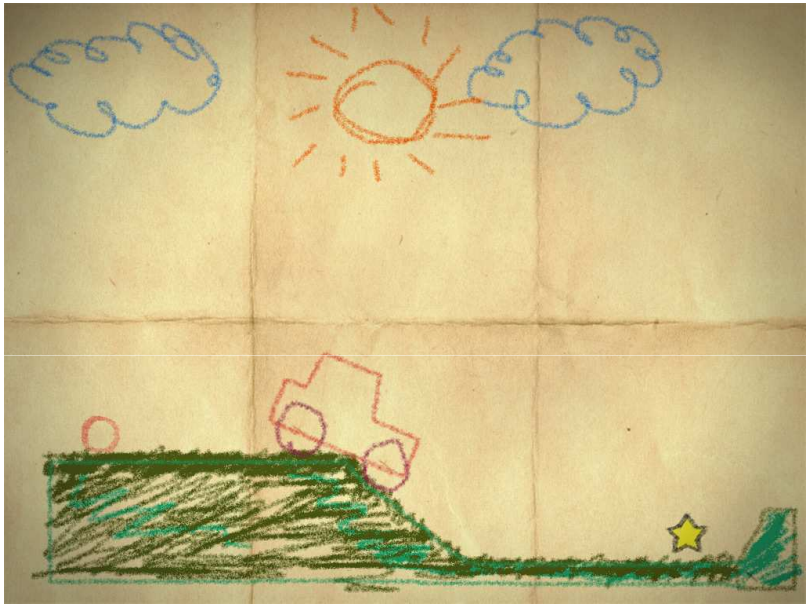


เกมส์

Crytek: Crysis

เกมส์





Kloonigames: Crayon Physics

ปัญหาสำคัญ

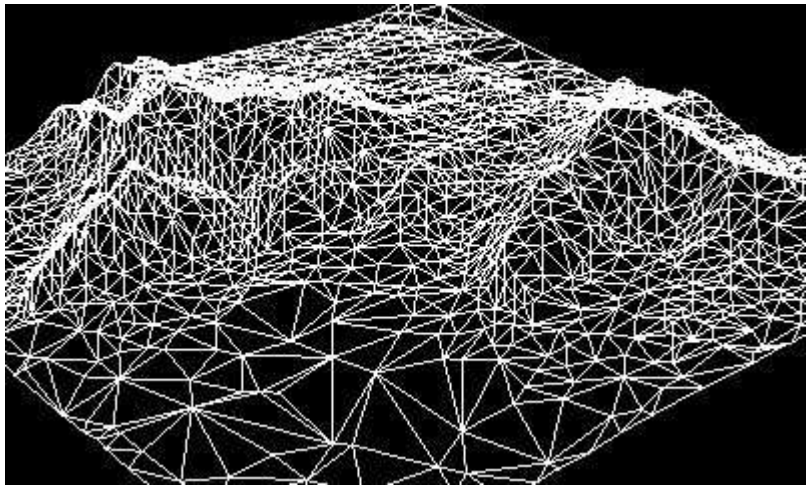
- การจัดการแบบจำลอง (modeling)
  - วิธีเก็บข้อมูลวัตถุจริงในคอมพิวเตอร์
- การให้แสงและเงา (rendering)
  - นำแบบจำลองมาสร้างเป็นรูปที่สวยงาม
- การจัดการความเคลื่อนไหว (animation)
  - วิธีสร้างและเก็บข้อมูลความเคลื่อนไหว
  - การจำลองปรากฏการณ์ธรรมชาติ

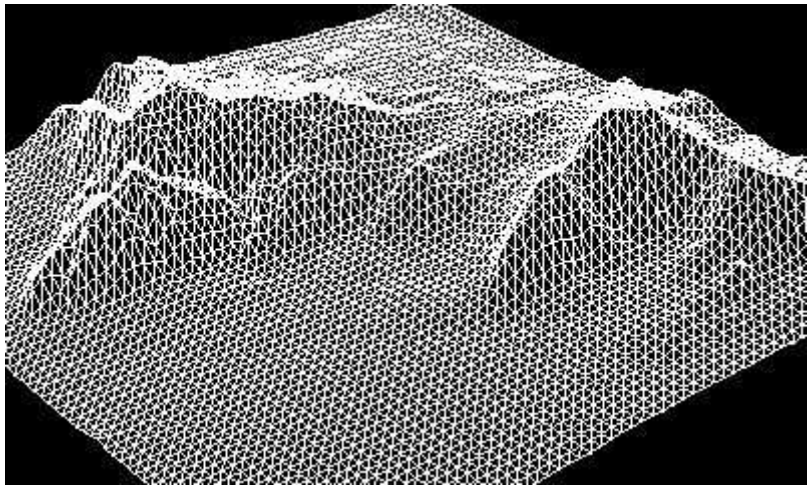
การจัดการแบบจำลอง

- แบบจำลองสำหรับ:
  - รูปร่าง รูปทรง
  - พื้นผิว
  - สมบัติการสะท้อนและดูดซับแสงของวัตถุ
- ปัญหา
  - เก็บข้อมูลอะไร?
  - จะดึงข้อมูลจากวัตถุจริงๆ ได้อย่างไร?

แบบจำลองรูปร่าง

- สามเหลี่ยมและหลายเหลี่ยม

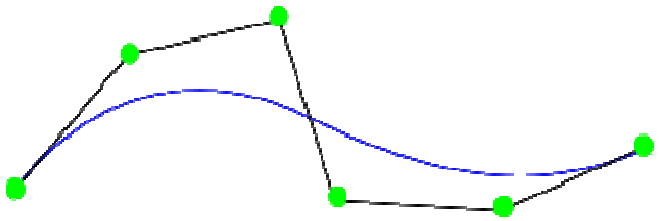


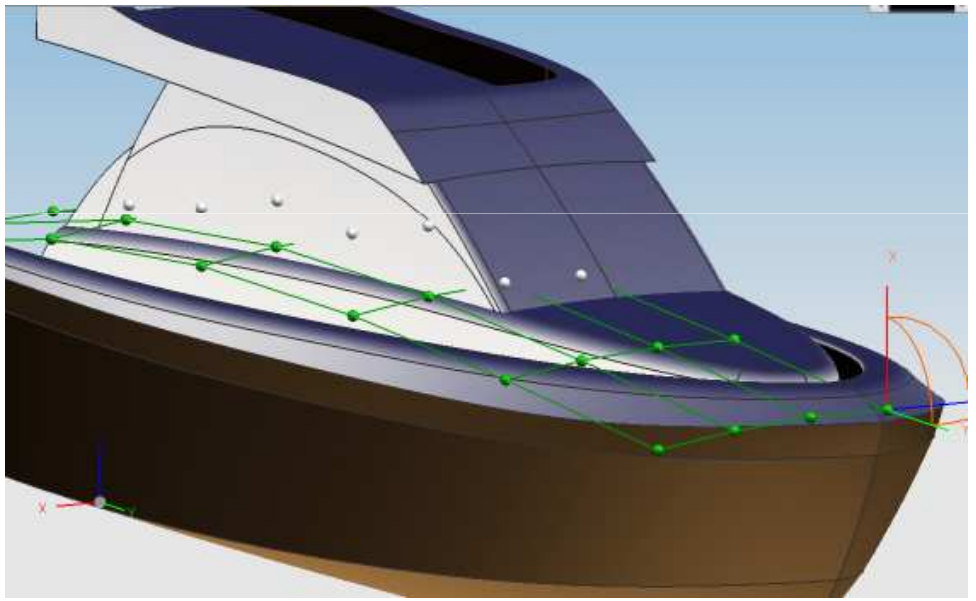


http://amber.rc.arizona.edu/dx/vtkDecimateDX.html

แบบจำลองรูปร่าง (ต่อ)

- เส้นโค้งพหุนาม

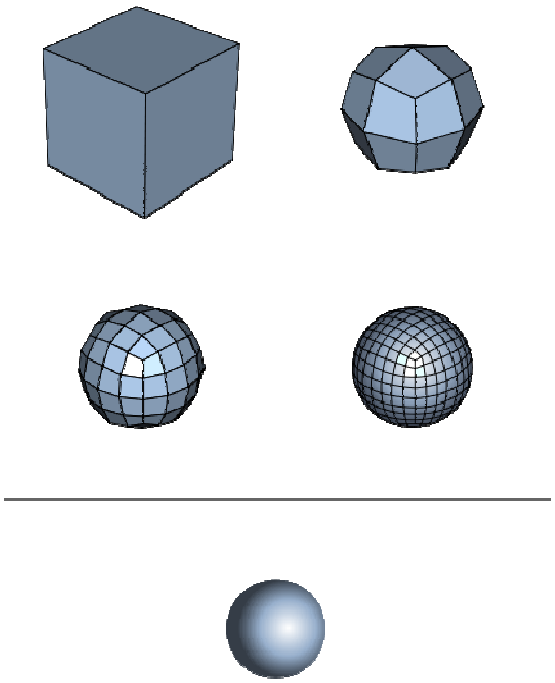




http://en.wikipedia.org/wiki/Nurbs

แบบจำลองรูปร่าง (ต่อ)

- Subdivision Surface



http://en.wikipedia.org/wiki/Subdivision_surface

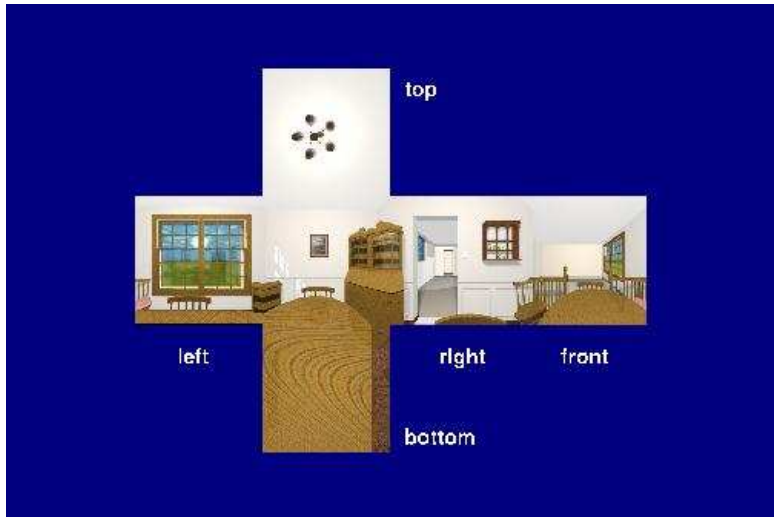
แบบจำลองรูปร่าง (ต่อ)

- Pixar’s Geri’s Game

http://www.pixar.com/shorts/gg/index.html

แบบจำลองพื้นผิว

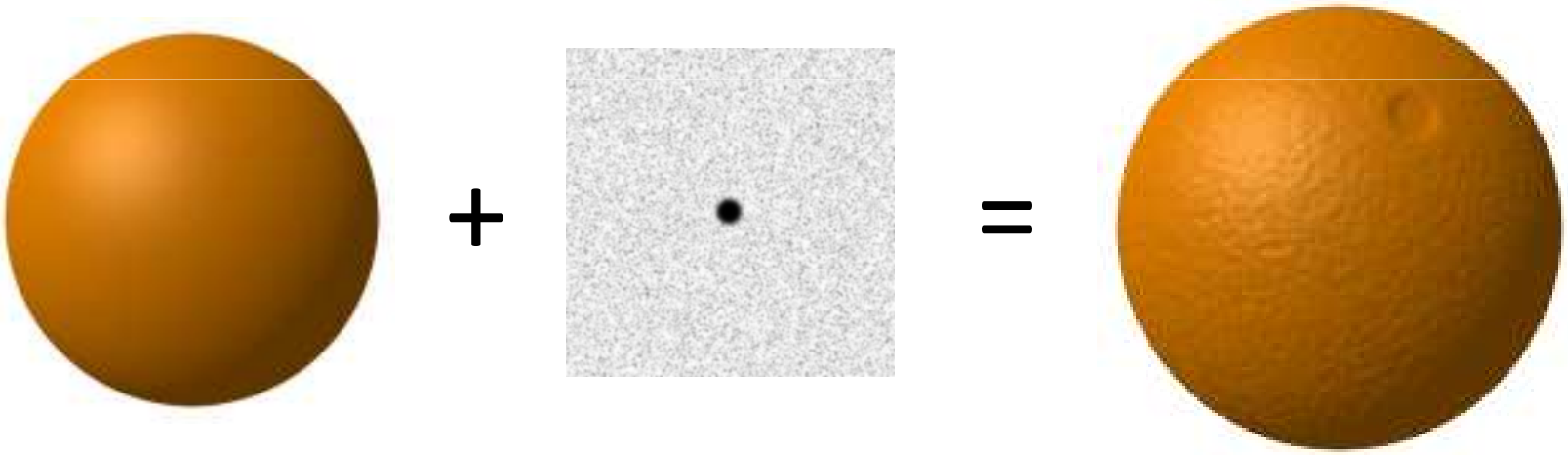
- Texture Mapping



http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/

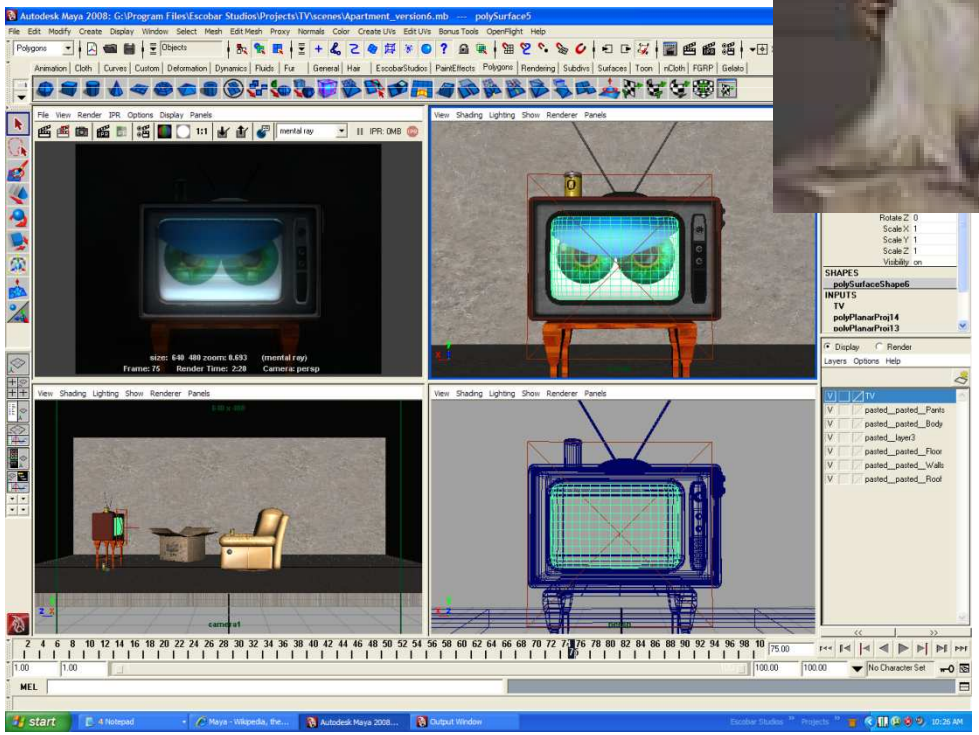
แบบจำลองพื้นผิว (ต่อ)

- Bump Mapping



การสร้างแบบจำลอง

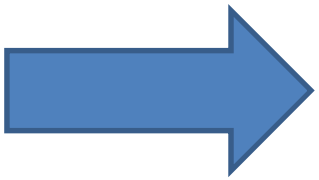
- สแกนเข้ามา
- สร้างเอาจากซอฟต์แวร์

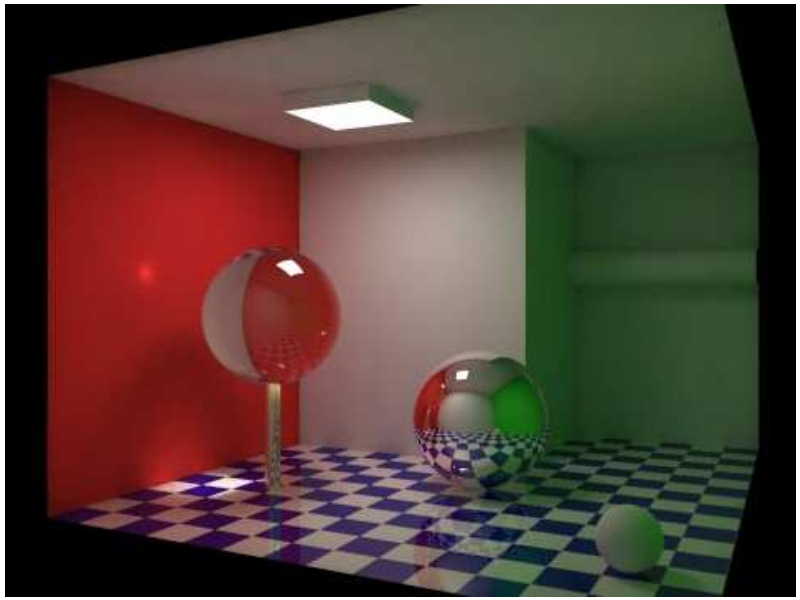


การให้แสงและเงา

- สร้างรูปภาพจากแบบจำลองทางคณิตศาสตร์

```
40.3765 246.3446 -13.3601
41.7488 226.0027 -5.0658
48.3294 235.3752 -7.3497
37.2949 230.1558 -9.6773
46.8526 239.2049 -10.7724
35.0925 232.2118 -10.9210
49.2234 231.9015 -5.4622
39.5274 227.7154 -6.8570
36.7923 240.2518 -18.0725
40.9546 241.5318 -16.3400
53.2942 227.1024 -17.4600
51.4157 231.8651 -20.9840
45.7685 234.6469 -25.0268
32.3952 239.7475 -5.4070
36.2495 235.5937 -5.3574
31.0568 236.1462 -9.5742
34.1015 253.4861 -8.2545
31.5805 251.6262 -9.3695
33.9048 256.8511 -4.1244
```





http://en.wikipedia.org/wiki/Global_illumination

Physically Based Rendering

- ให้แสงเงาให้สมจริงตามหลักฟิสิกส์

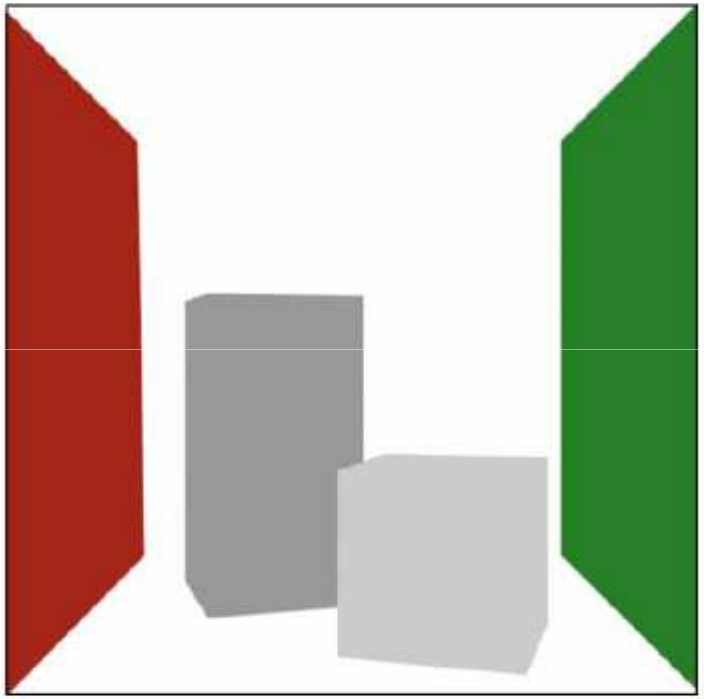
http://en.wikipedia.org/wiki/Rendering

Non-Photorealistic Rendering

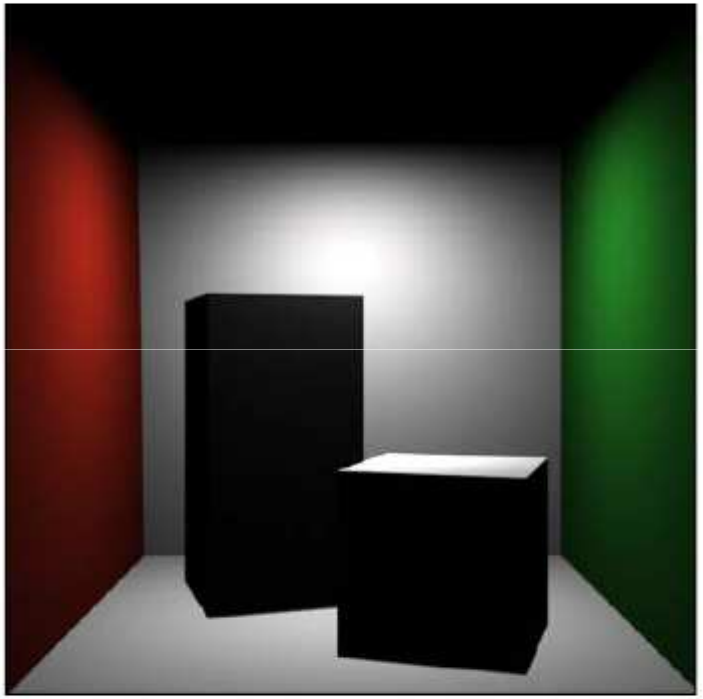
- ให้สีไม่ตรงกับความเป็นจริงเพื่อความสวยงาม

The Legend of Zelda: The Wind Waker
http://en.wikipedia.org/wiki/Toon_shading

Lighting: Diffuse Reflection

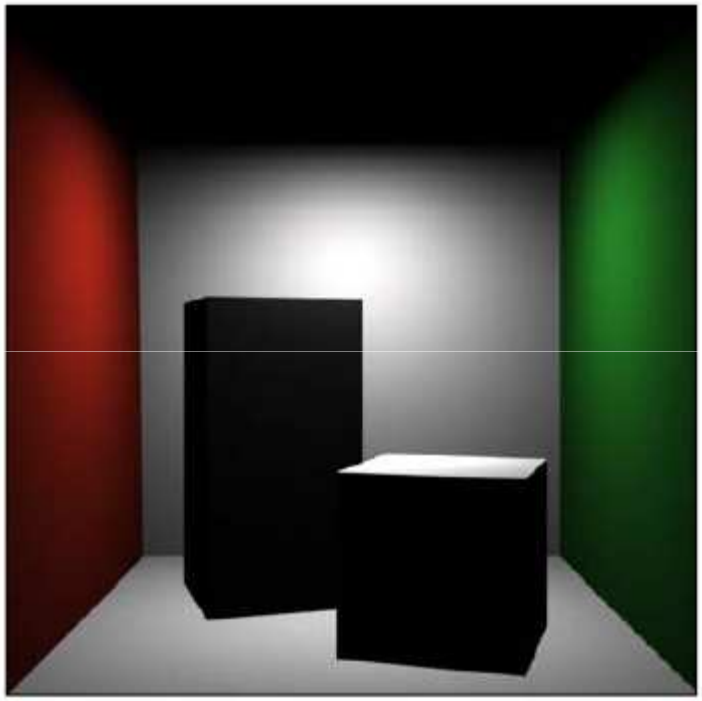


Surface Color

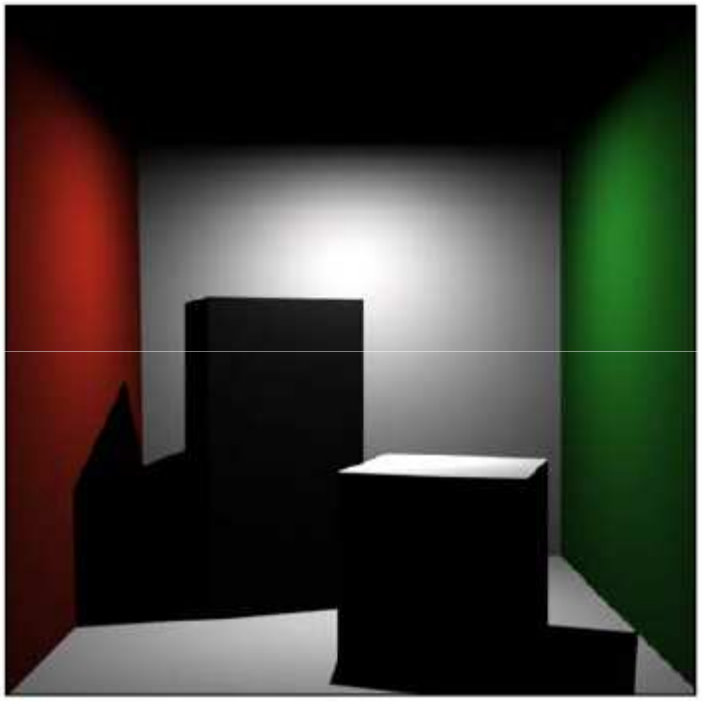


Diffuse Shading
Point Light Source

Lighting: Shadows

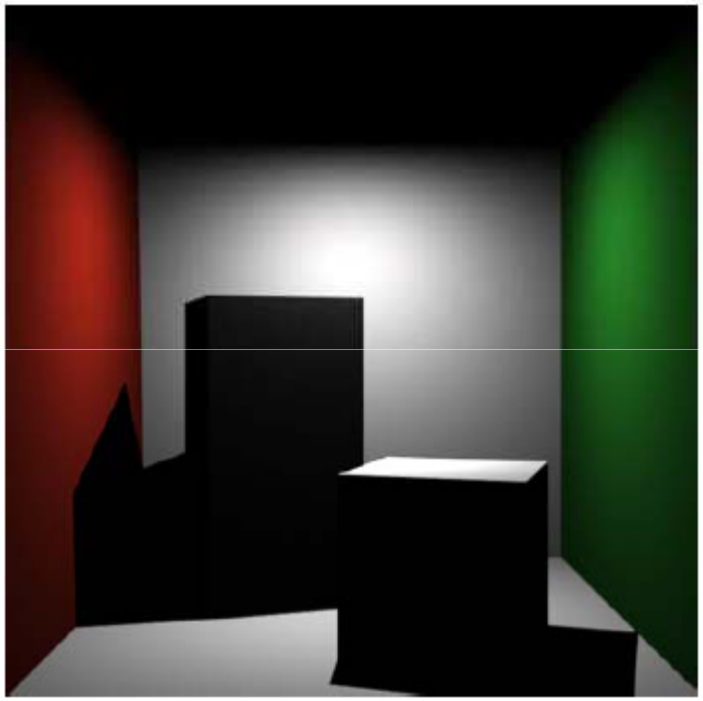


No Shadows
Point Light Source



Shadows
Point Light Source

Lighting: Soft Shadows

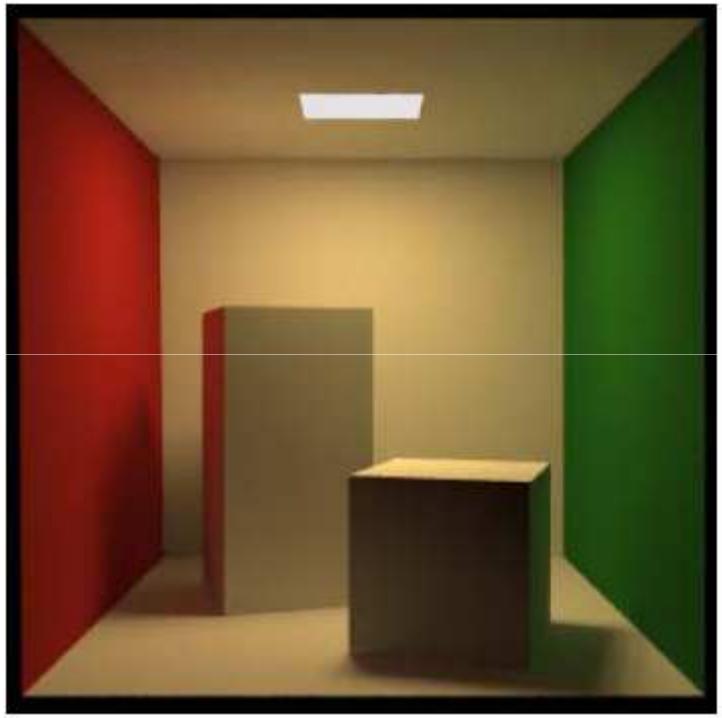


Hard Shadows
Point Light Source

Soft Shadows
Area Light Source

Lighting: Radiosity

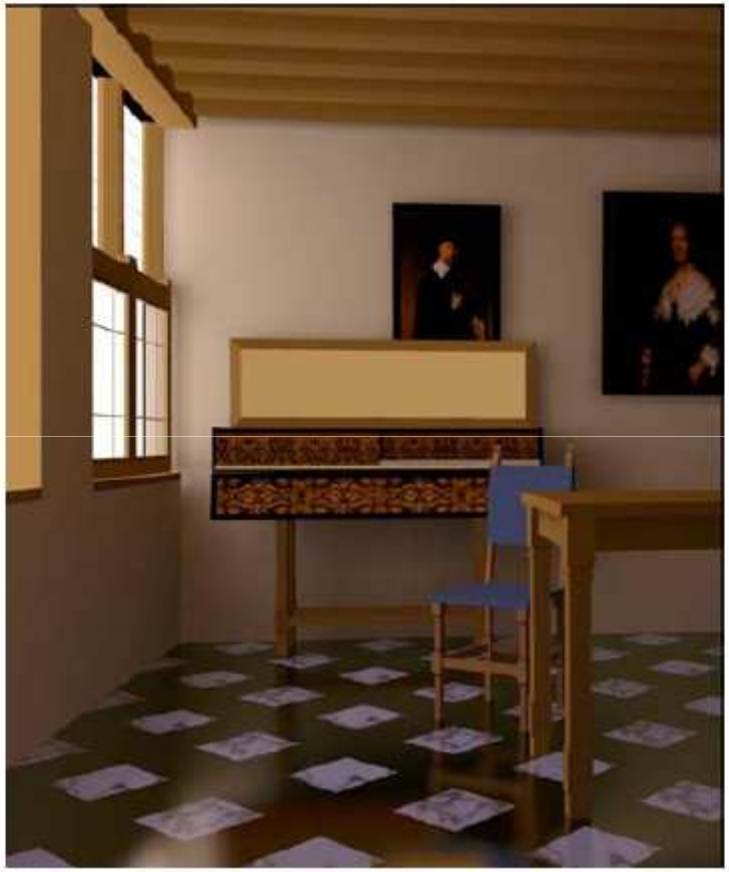
Soft Shadows
Area Light Source



Inter-reflection, Diffuse)
Area Light Source

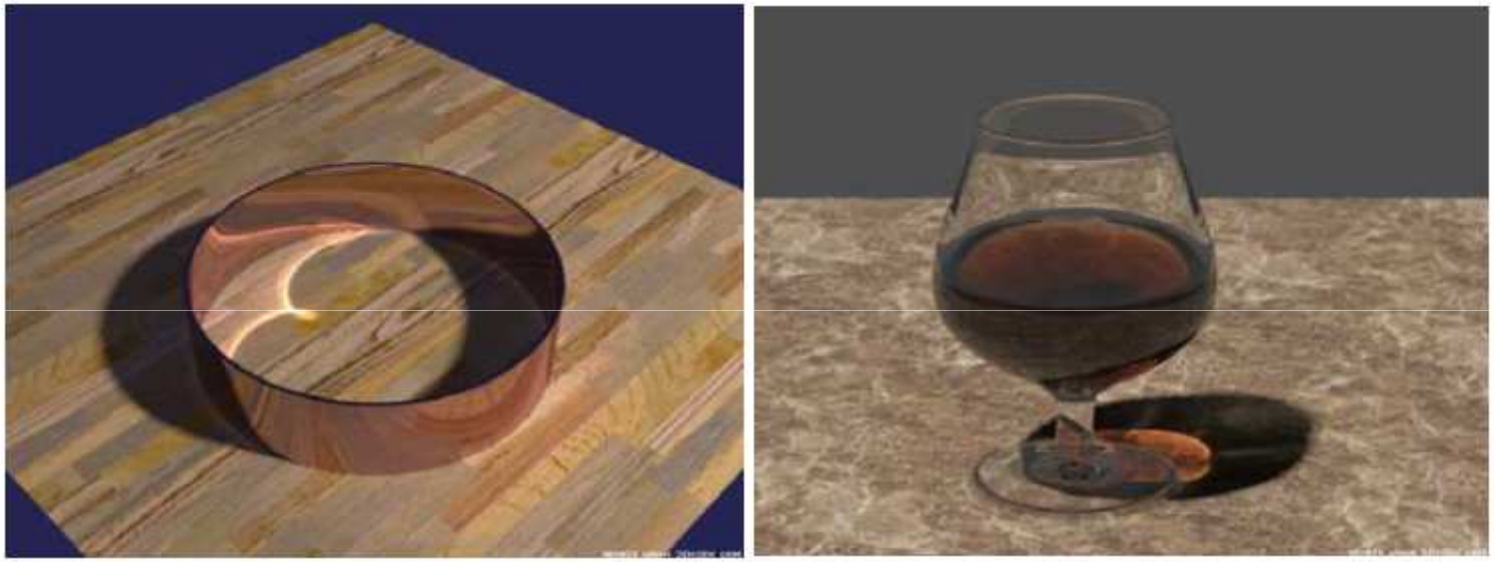
Early Radiosity

Early Diffuse+Glossy



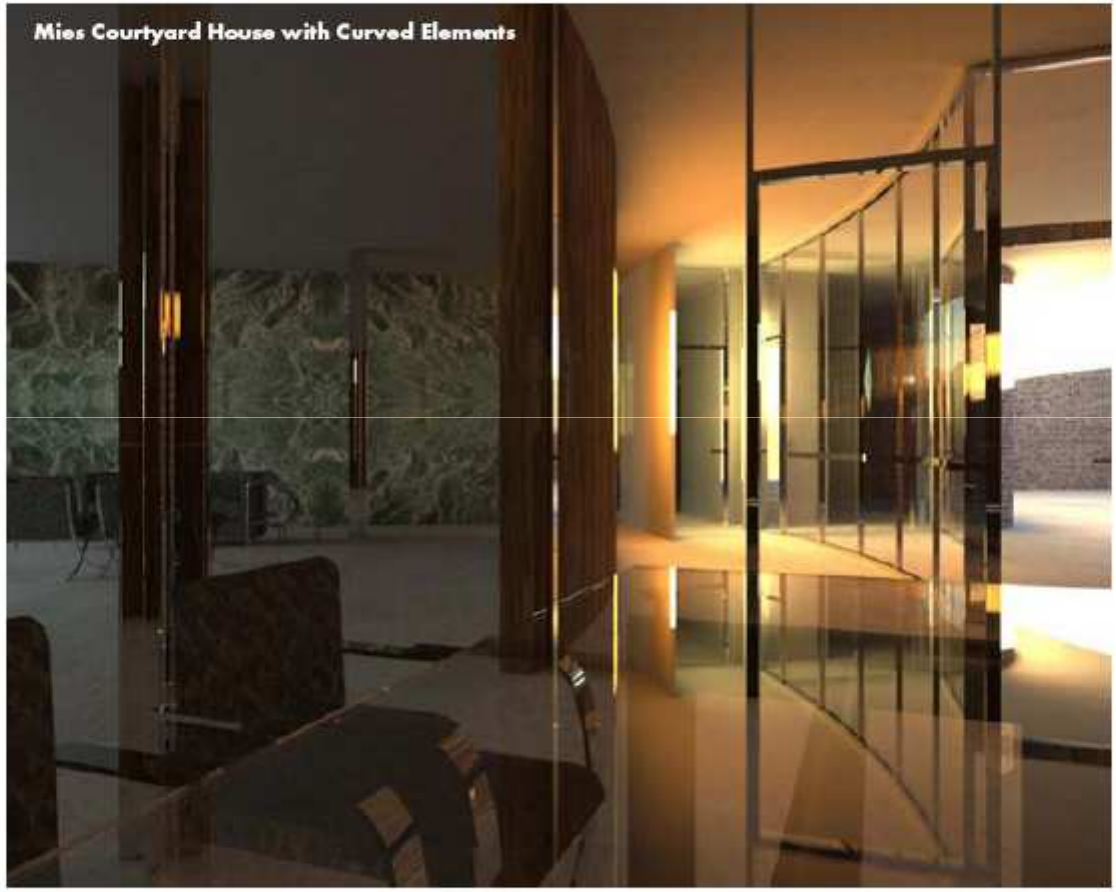
Tribute to Vermeer
Program of Computer Graphics, Cornell

Caustics



Jensen 1995

Complex Indirect Illumination



Modeling: Stephen Duck; Rendering: Henrik Wann Jensen

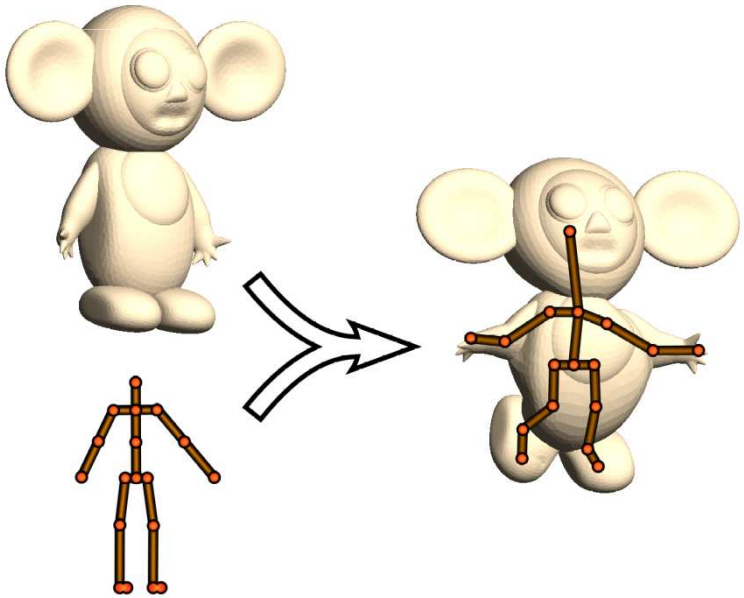
Translucency

Surface Reflection

Subsurface Reflection

การจัดการความเคลื่อนไหว

- ข้อมูลความเคลื่อนไหวหน้าตาเป็นอย่างไร?
  - โครงกระดูก
  - การหมุนของข้อ



Ilya Baran and Jovan Popovic. *Automatic Rigging and Animation of 3D Characters.* SIGGRAPH 2007.

การจัดการความเคลื่อนไหว

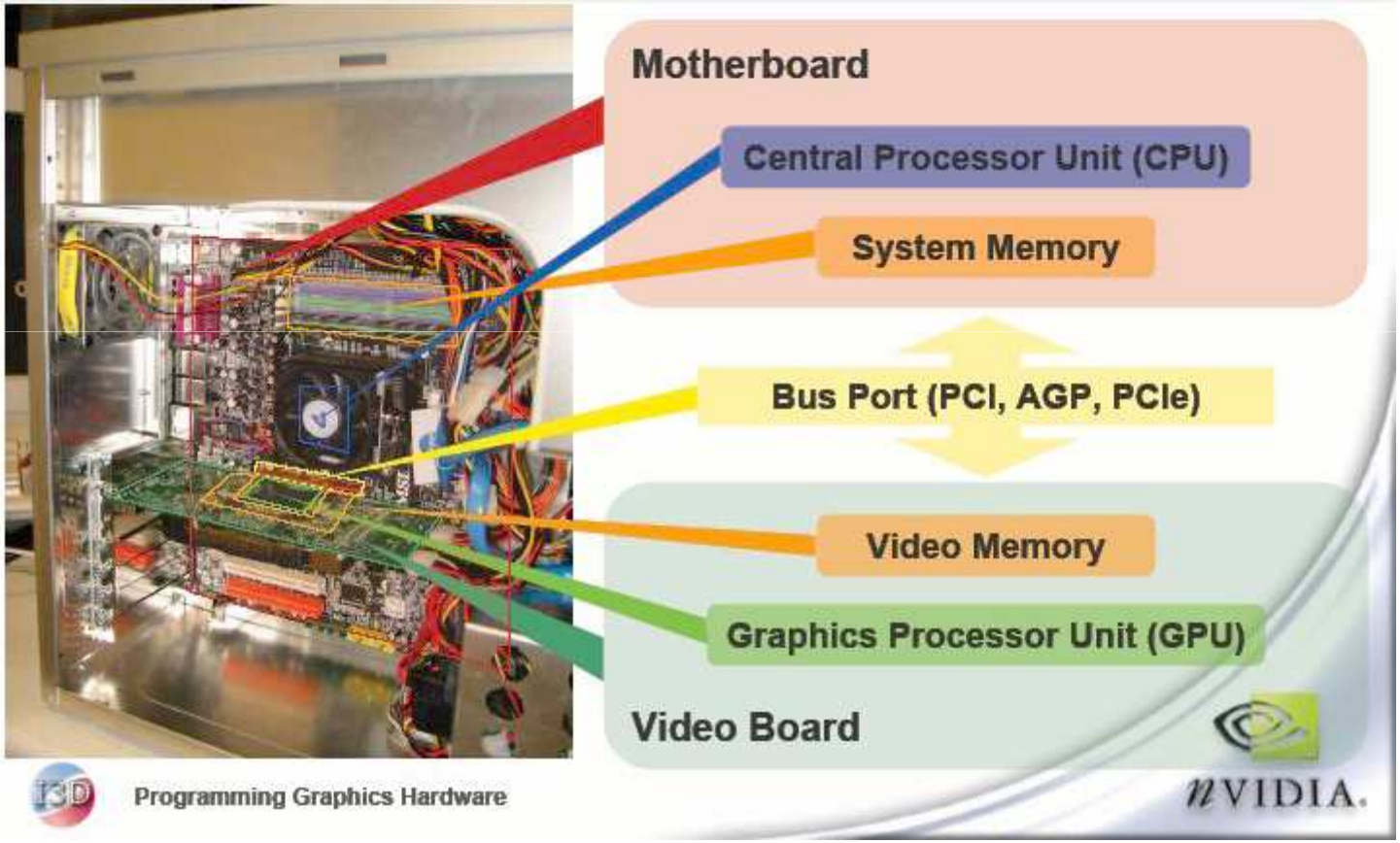
- เก็บข้อมูลอย่างไร: Motion capture

การจำลองปรากฏการณ์ธรรมชาติ

- จำลองอะไร?
  - ไฟ, น้ำ, ผ้า, ของแตก, ของนุ่ม, ควัน, ฯลฯ
- แก้สมการเชิงอนุพันธ์
- ไม่ต้องถูกต้อง 100% แค่สวยก็พอ

ระบบคอมพิวเตอร์กราฟิกส์
ในเครื่องคอมพิวเตอร์ส่วนบุคคล

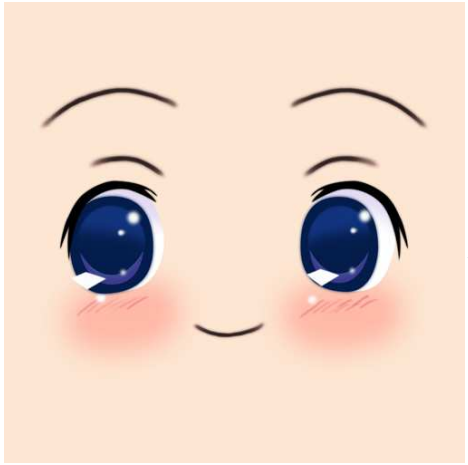
ในเครื่องคอมของคุณ...



รูปที่แล้วมีอะไรบ้าง?

- CPU, Memory, Bus
  - เรารู้ดีอยู่แล้วว่ามันทำอะไร
- GPU
  - ทำงานเกี่ยวกับกราฟิกส์
  - ข้อมูลเข้า
    - โมเดลรูปทรง (ตำแหน่งของจุด ความเชื่อมโยงกันของจุด และสีของจุด)
    - จิตรกรรมฝาผนัง
  - ข้อมูลออก
    - รูปบนหน้าจอ

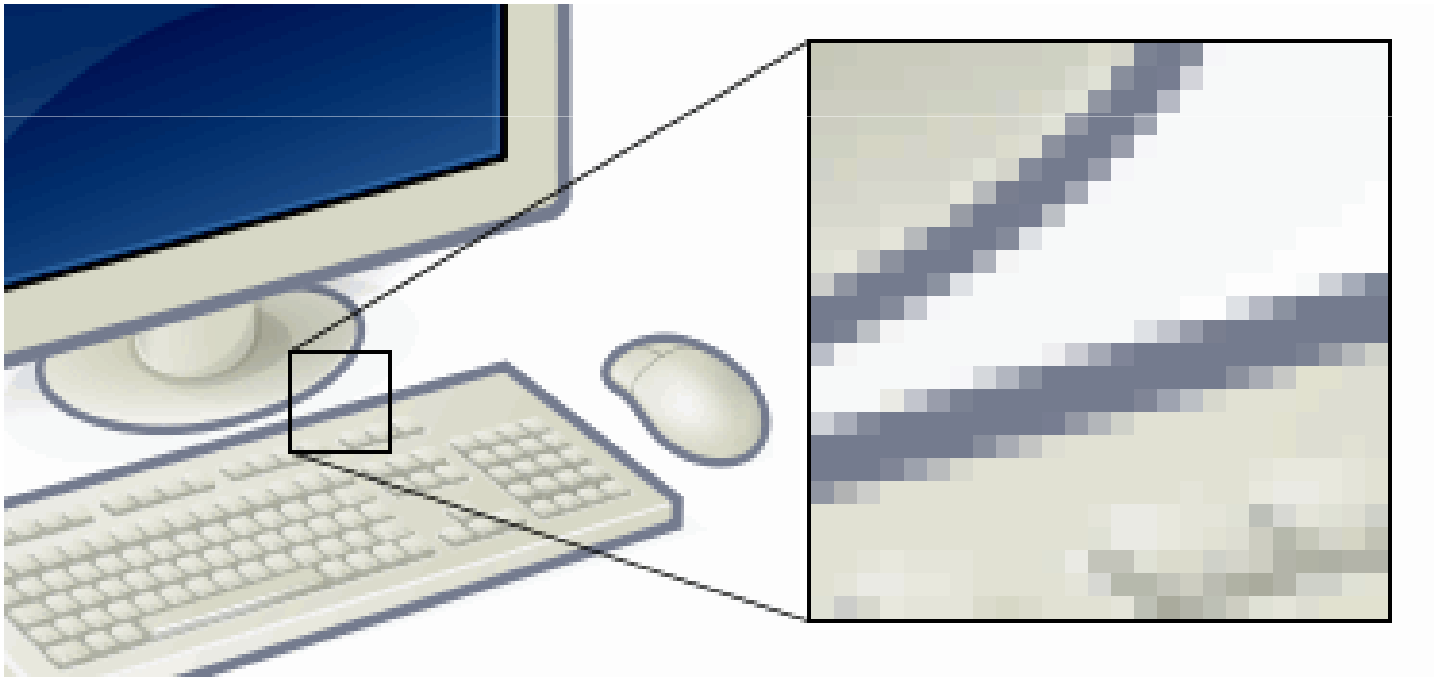
40.3765 246.3446 -13.3601
41.7488 226.0027 -5.0658
48.3294 235.3752 -7.3497
37.2949 230.1558 -9.6773
46.8526 239.2049 -10.7724
35.0925 232.2118 -10.9210
49.2234 231.9015 -5.4622
39.5274 227.7154 -6.8570
36.7923 240.2518 -18.0725
40.9546 241.5318 -16.3400
53.2942 227.1024 -17.4600
51.4157 231.8651 -20.9840
45.7685 234.6469 -25.0268
32.3952 239.7475 -5.4070
36.2495 235.5937 -5.3574
31.0568 236.1462 -9.5742
34.1015 253.4861 -8.2545
31.5805 251.6262 -9.3695
33.9048 256.8511 -4.1244



GPU

ภาพ

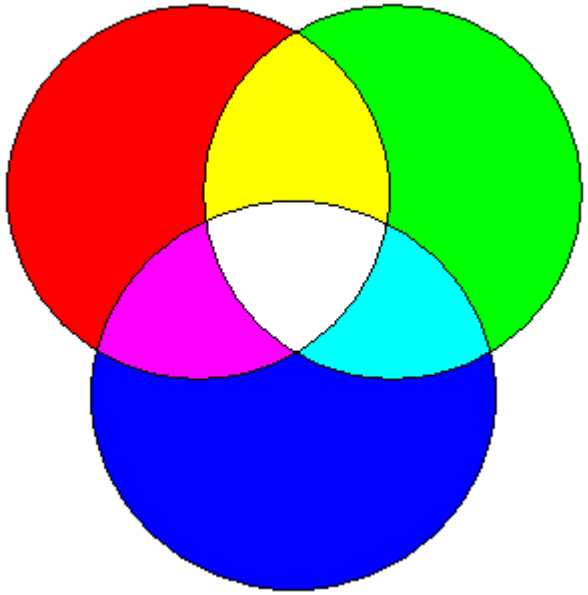
- ตารางสี่เหลี่ยมผืนผ้า แต่ละช่องมีสีหนึ่งสี
- แต่ละช่องเรียกว่า พิกเซล (pixel)



http://en.wikipedia.org/wiki/Pixels

สี

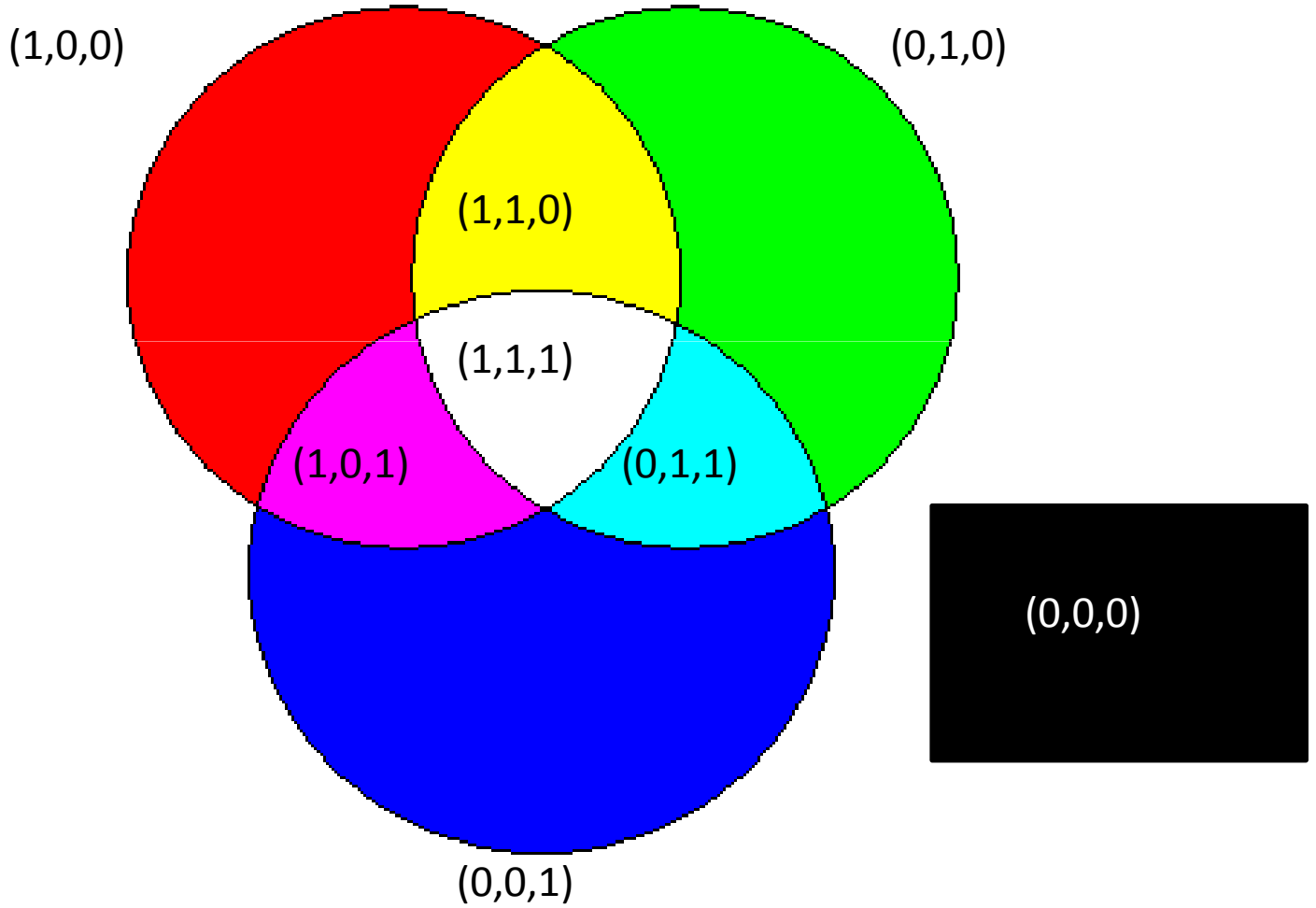
- สี = เวกเตอร์ (R, G, B) เลขแต่ละตัวมีค่าตั้งแต่ 0 ถึง 1
  - R บอกระดับความเข้มของแสงสีแดง
  - G บอกระดับความเข้มของแสงสีเขียว
  - B บอกระดับความเข้มของแสงสีน้ำเงิน



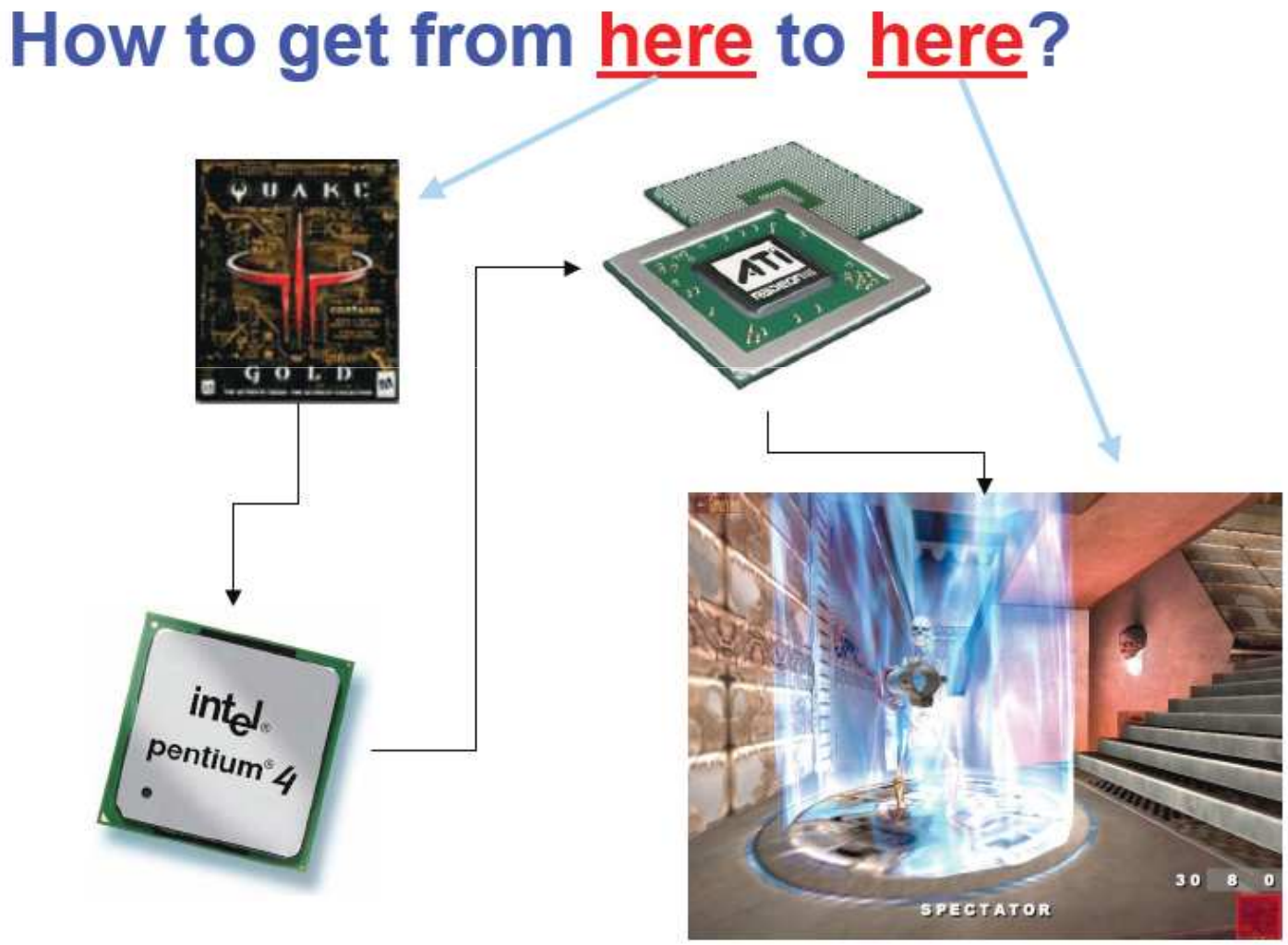
Trichromatic Theory of Vision

- สีที่มนุษย์มองเห็นแบ่งออกเป็นสามส่วน
  - แดง เขียว น้ำเงิน
  - ประสาทสัมผัสของมนุษย์ของแต่ละสีเป็นอิสระจากกัน
  - สีอื่นๆ เกิดจาก การนำสีทั้งสามนี้มาประกอบกัน
- หลักฐาน
  - เซลล์โคนในเรตินามีสามชนิด
  - แต่ละชนิดไวต่อ สีแดง สีเขียว สีน้ำเงิน ตามลำดับ

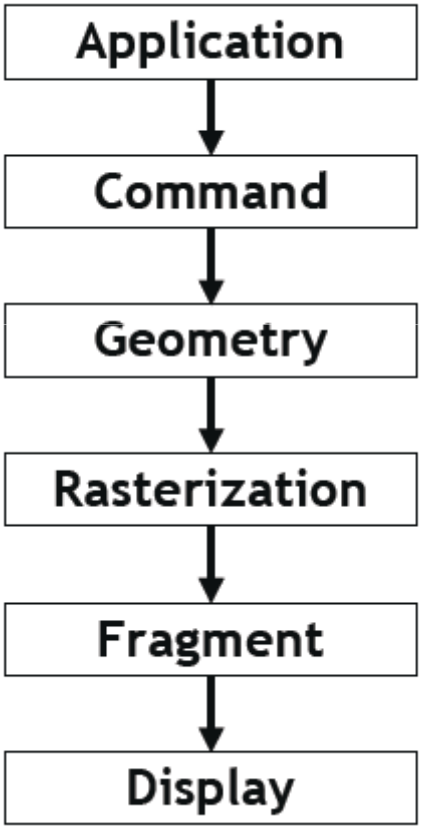
สีหลักๆ



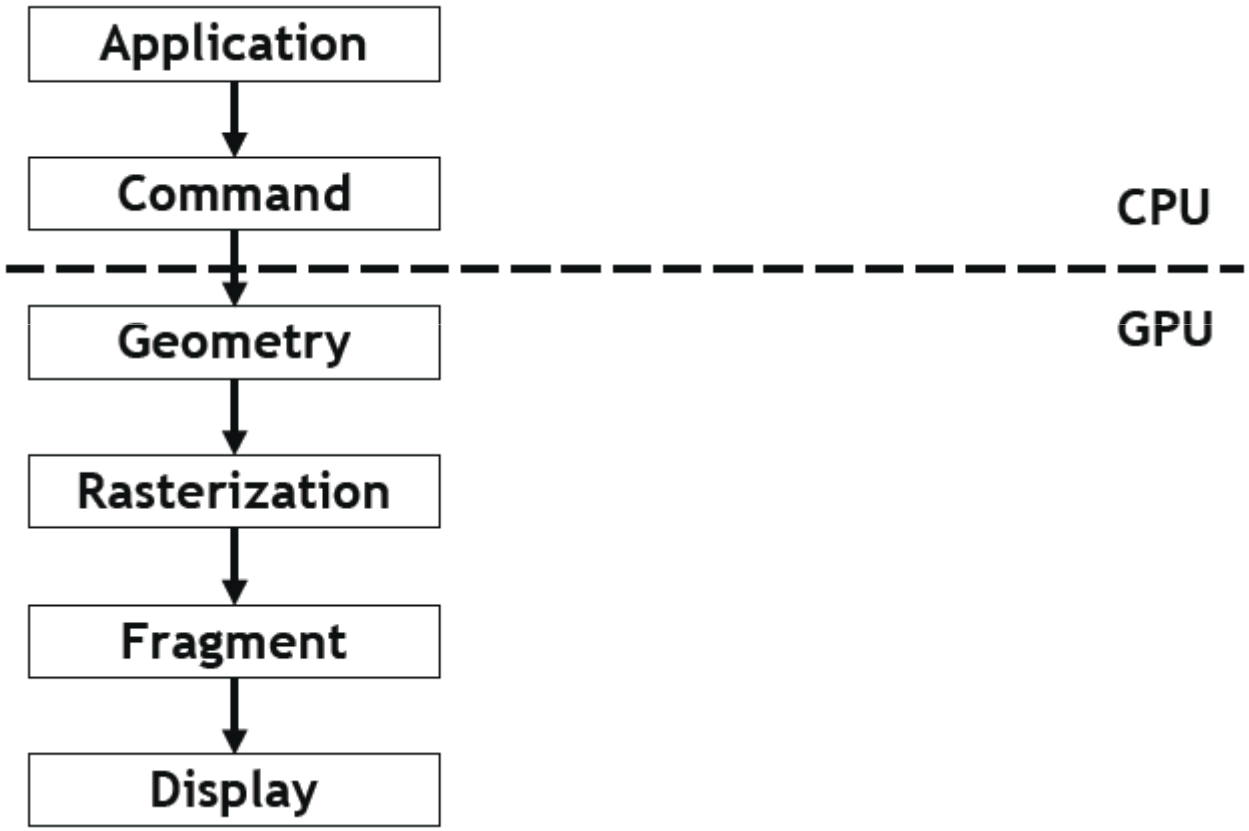
Graphics Pipeline



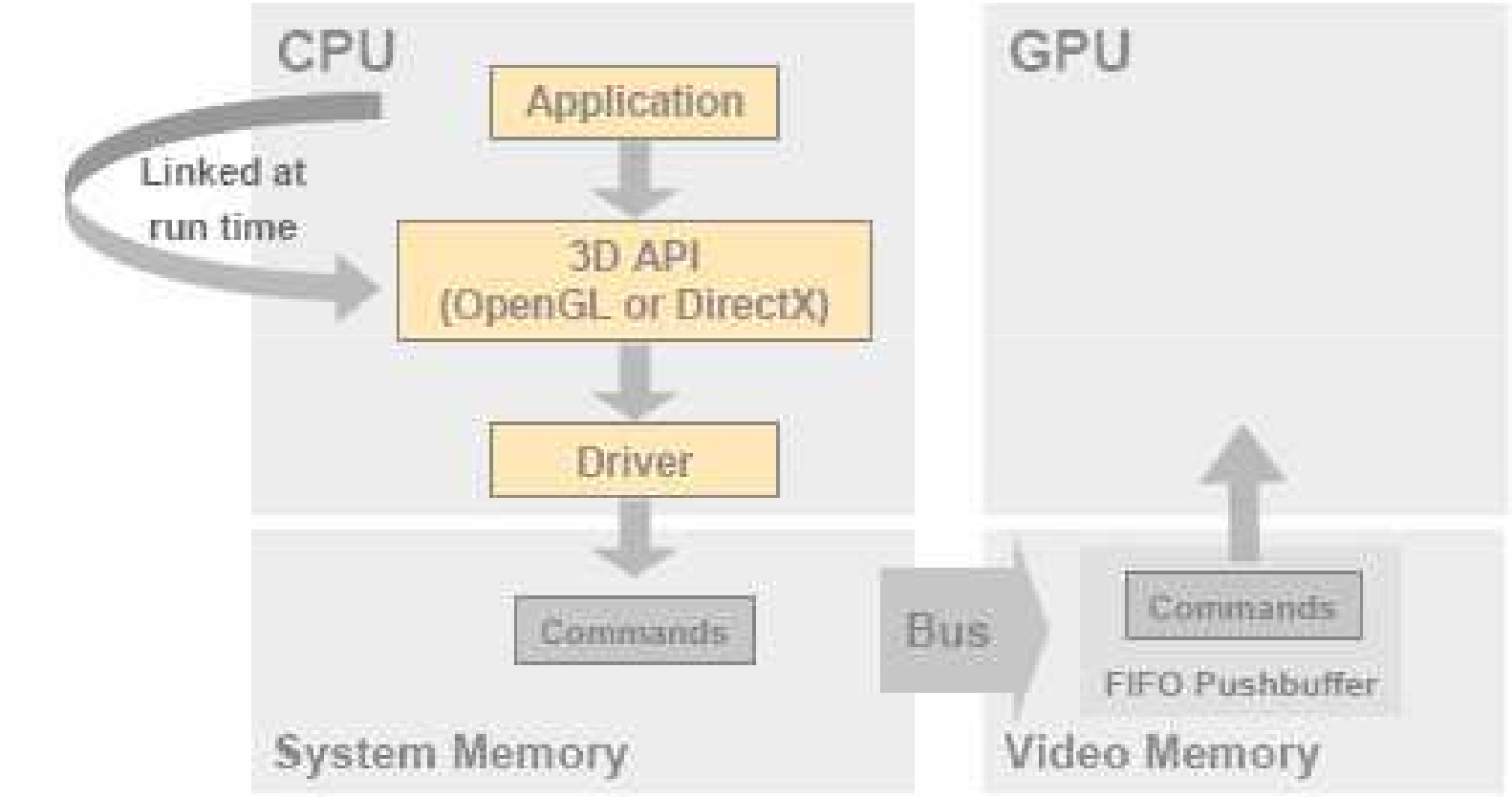
Graphics Pipeline (ต่อ)



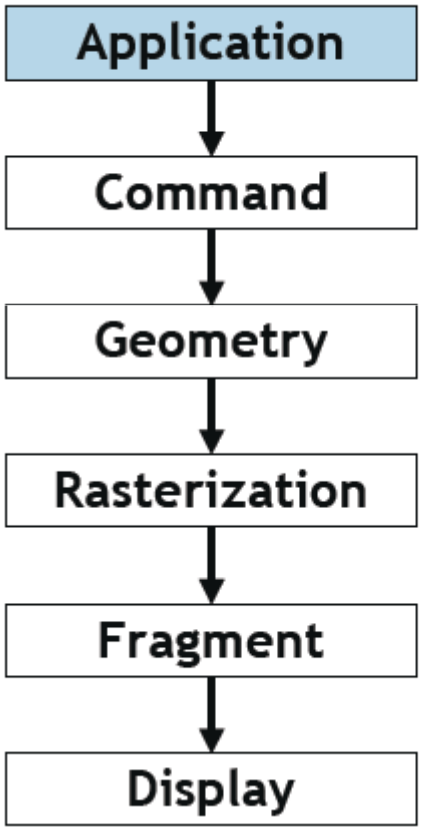
Graphics Pipeline (ต่อ)



Graphics Pipeline (ต่อ)

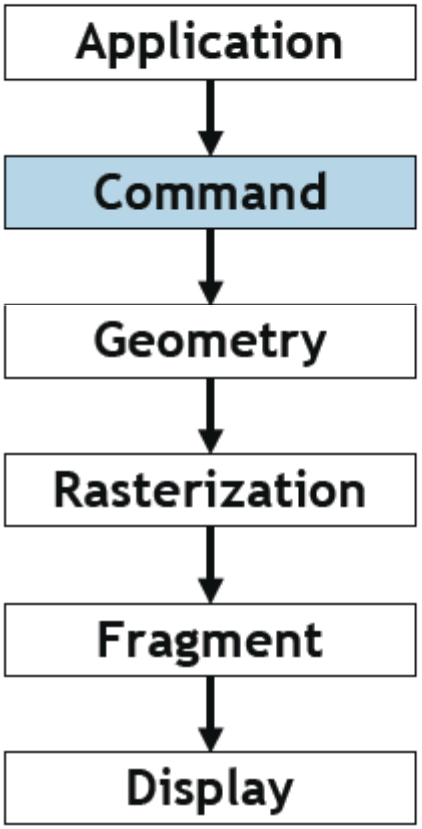


Graphics Pipeline (ต่อ)



- Quake 3:
- define game behavior
- networking
- user input events
- sound processing
- game AI
- game physics

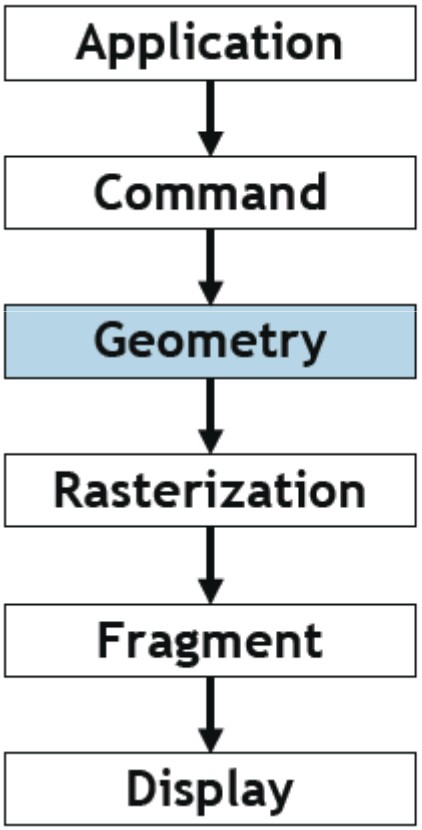
Graphics Pipeline (ต่อ)



- Quake 3:
- send OpenGL commands

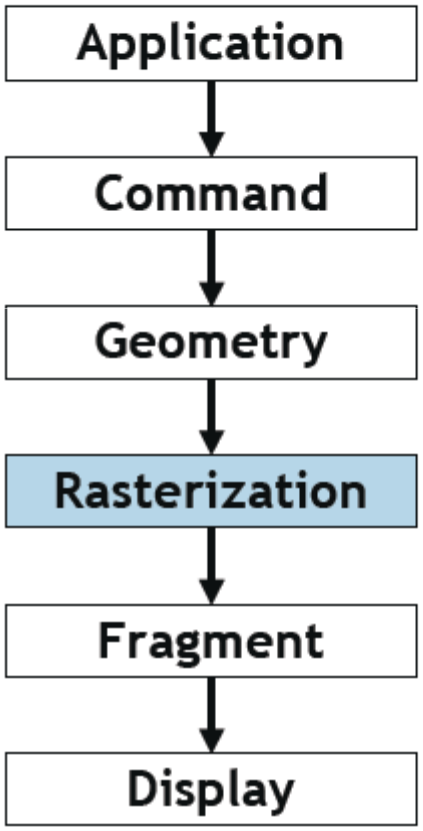
- OpenGL driver:
- process GL command stream
- talk to GPU

Graphics Pipeline (ต่อ)



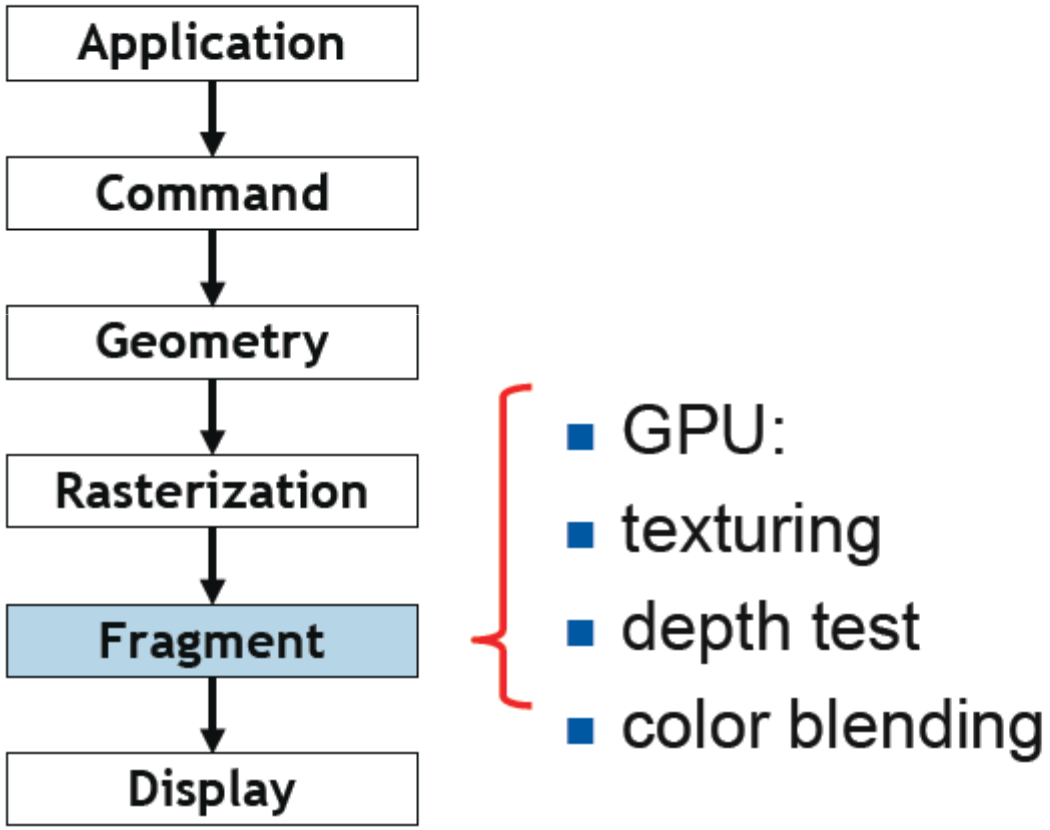
- GPU:
- vertex transformations
- vertex lighting
- clipping
- primitive assembly

Graphics Pipeline (ต่อ)

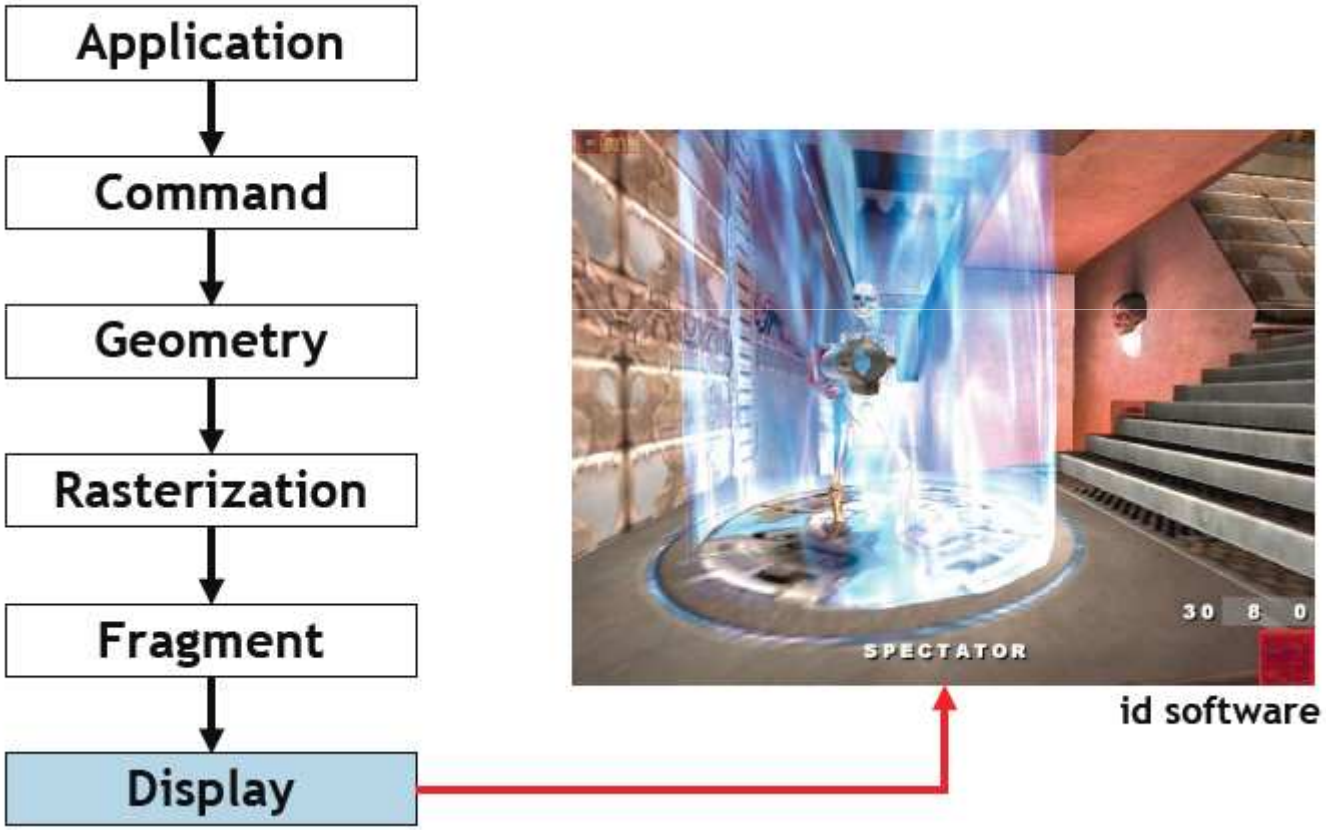


- GPU:
- convert triangles to fragments
- tex coordinate interpolation
- color interpolation

Graphics Pipeline (ต่อ)



Graphics Pipeline (ต่อ)

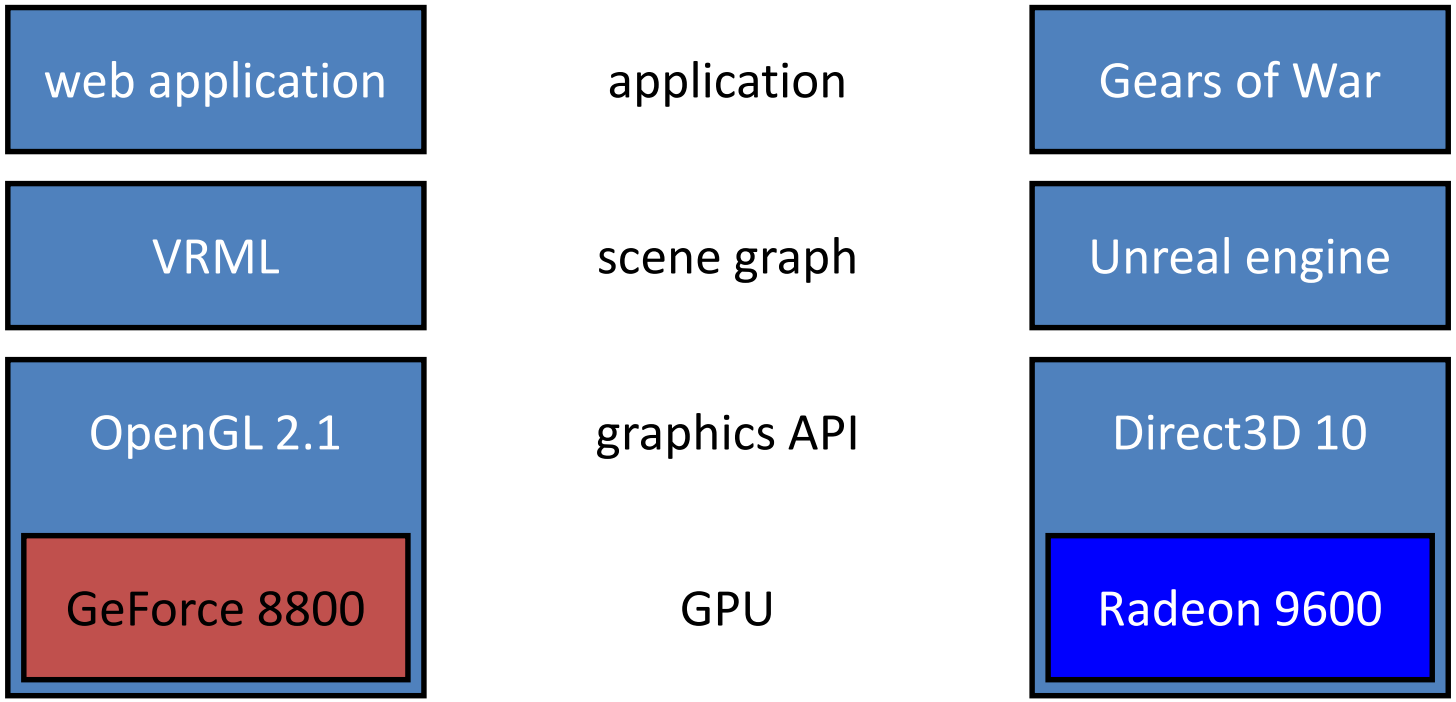


OPENGL และ GLUT

OpenGL

- Application Programming Interface (API) สำหรับควบคุม GPU
- ผู้ใช้ OpenGL ระบุรูปทรงและรูปร่างพื้นฐาน (จุด เส้น และรูปหลายเหลี่ยม) ผ่านทาง OpenGL
- OpenGL จะทำหน้าที่สร้างภาพไว้บน framebuffer ให้
- ใช้สร้างโปรแกรมที่มีการตอบสนองต่อผู้ใช้แบบทันทีทันควัน (interactive) และโปรแกรมที่มีภาพเคลื่อนไหว
- ทำหน้าที่เดียวกับ Direct3D และเป็นคู่แข่งทางการค้ากันอยู่

กายวิภาคของโปรแกรมทางคอมพิวเตอร์กราฟฟิกส์



คำศัพท์

- Bitplane
  - เนื้อที่ในหน่วยความจำที่เก็บข้อมูล 1 บิทของทุกพิกเซลที่อยู่บนจอภาพ
- Framebuffer
  - Bitplane หลายๆ bitplane ที่เก็บข้อมูลทั้งหมดที่ใช้ควบคุมหน้าจอ
- Buffer
  - Bitplane กลุ่มหนึ่งที่ใช้เก็บข้อมูลบางอย่าง
- Application Programming Interface (API)
  - ฟังก์ชันและ object อื่นๆ ในภาษาระดับสูงที่ให้โปรแกรมประยุกต์ใช้สำหรับติดต่อกับระบบฮาร์ดแวร์หรือซอฟต์แวร์ต่างๆ

สิ่งที่ OpenGL ไม่ทำ

- จัดการการติดต่อกับผู้ใช้
- จัดการวินโดวส์
- วาดและจัดการรูปทรงที่ซับซ้อน เช่น รถถัง ต้นไม้ ฯลฯ
  - ถึงแม้ว่าคุณจะสามารถใช้รูปทรงง่ายๆ ของ OpenGL สร้างมันได้ก็ตาม
  - ส่วนใหญ่คุณต้องเขียน library ขึ้นมาจัดการกับพวกนี้เอง
- จัดการ framebuffer
  - เป็นความรับผิดชอบของคุณที่ต้องเตรียม framebuffer ให้ OpenGL

GLUT

- OpenGL Utility Toolkit
- ใช้สำหรับจัดการการติดต่อกับผู้ใช้และจัดการวินโดวส์
  - ทำสิ่งที่ OpenGL ไม่ทำ
- เอาไปใช้เขียนโปรแกรมประยุกต์จริงๆ คงยาก
  - ไม่มี GUI Widget ให้ใช้เลย
  - ต้องรับข้อมูลจากผู้ใช้ตามที่ GLUT กำหนด
- แต่ทำให้การเรียนรู้ OpenGL ง่ายขึ้นมาก

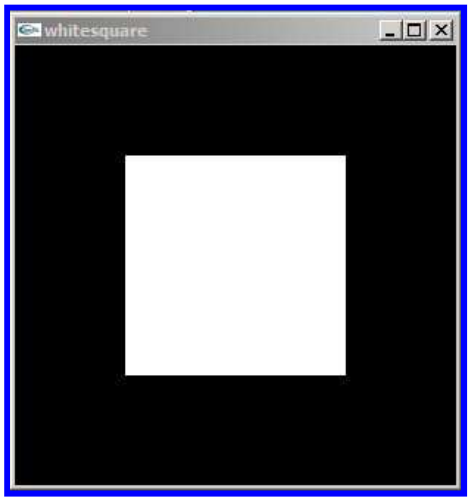
ตัวอย่าง

OpenGL

```
void draw() {
    glClearColor(0.0, 0.0, 0.0, 0.0)
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_POLYGON);
        glVertex3f(-0.5, -0.5, 0.0);
        glVertex3f( 0.5, -0.5, 0.0);
        glVertex3f( 0.5,  0.5, 0.0);
        glVertex3f(-0.5,  0.5, 0.0);
    glEnd();
    glFlush();
}
```

GLUT

```
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutCreateWindow("whitesquare");
    glutDisplayFunc(draw);
    glutMainLoop();
}
```



เฉพาะส่วนของ OpenGL

```
glClearColor(0.0, 0.0, 0.0, 0.0)
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0, 1.0, 1.0);
glBegin(GL_POLYGON);
  glVertex3f(-0.5, -0.5, 0.0);
  glVertex3f( 0.5, -0.5, 0.0);
  glVertex3f( 0.5,  0.5, 0.0);
  glVertex3f(-0.5,  0.5, 0.0);
glEnd();
glFlush();
```

ทีละคำสั่ง

- glClearColor(0.0, 0.0, 0.0, 0.0)
  - เปลี่ยนสีที่จะใช้ล้างหน้าจอเป็นสีดำ
- glClear(GL_COLOR_BUFFER_BIT)
  - ล้าง bitplane ที่เก็บสีด้วยสีทีกำหนดใน glClearColor
- glColor3f(1.0, 1.0, 1.0)
  - เปลี่ยนสีเป็นสีขาว
  - จุดที่วาดต่อจากนี้ไปจะเป็นสีขาว

ทีละคำสั่ง (ต่อ)

- glBegin(GL_POLYGON)
  - บอกว่าต่อไปเราจะวาดรูปหลายเหลี่ยม
- glVertex3f(x, y, z)
  - กำหนดจุด
- glEnd()
  - บอกว่าสิ่งที่เริ่มไปตั้งแต่ glBegin ที่แล้วได้เสร็จสิ้นแล้ว
  - ในที่นี้คือบอกว่ากำหนดรูปหลายเหลี่ยมเสร็จแล้ว
- glFlush()
  - ทำให้คำสั่ง OpenGL ที่เคยสั่งมาถูกนำไปปฏิบัติงาน แทนที่จะถูกเก็บไว้ในหน่วยความจำเพื่อรอคำสั่งอื่น

คำสั่ง OpenGL

- ขึ้นต้นด้วย gl
- ตามด้วยชื่อคำสั่ง เช่น Vertex หรือ Color
- บางคำสั่งอาจมี*จำนวนและชนิด*ของ argument
  - 3f บอกว่าต้องการ argument เป็น float 3 ตัว
    - glVertex3f(1.0f, 3.0f, 4.0f);
  - 2i บอกว่าต้องการ argument เป็น int 2 ตัว
    - glVertex2i(-1, 5);
  - 3fv บอกว่าต้องการ argument เป็น pointer ไปยัง float 3 ตัว
    - float colorArray[] = {1.0f, 0.0f, 0.0f}
    - glColor3fv(colorArray);

ชนิดของ argument ในชื่อคำสั่ง

Suffix	Data Type	Typical Corresponding C-Language Type	OpenGL Type Definition
b	8-bit integer	signed char	GLbyte
s	16-bit integer	short	GLshort
i	32-bit integer	long	GLint, GLsizei
f	32-bit floating-point	float	GLfloat, GLclampf
d	64-bit floating-point	double	GLdouble, GLclampd
ub	8-bit unsigned integer	unsigned char	GLubyte, GLboolean
us	16-bit unsigned integer	unsigned short	GLushort
ui	32-bit unsigned integer	unsigned long	GLuint, GLenum, GLbitfield

OpenGL เป็น State Machine

- OpenGL จะจำค่าต่างๆ ที่ผู้ใช้กำหนดได้เอาไว้
- เมื่อผู้ใช้กำหนดค่า ค่านั้นจะถูกใช้ต่อไปเรื่อยๆ จนกว่าจะเปลี่ยน
- ค่าที่จำไว้ เช่น
  - สีที่ใช้ล้างหน้าจอ
  - สีของจุด
  - ทิศทางและตำแหน่งของกล้องถ่ายรูป
- ยกตัวอย่างเช่น เวลาเราเรียก glColor3f(1,1,1) แล้วสีของจุดที่กำหนดด้วย glVertex จะเป็นสีขาวไปจนกว่าจะเรียก glColor ใหม่อีกครั้ง

โค้ดตัวอย่างเฉพาะส่วนของ GLUT

```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutCreateWindow("whitesquare");
    glutDisplayFunc(draw);
    glutMainLoop();
}
```

glutInit

- glutInit(int *argc, char **argv)
  - ทำการตั้งค่าเริ่มต้นหลายๆ ค่าของ GLUT
  - สิ่งที่ต้องส่งให้คือ pointer ไปยังจำนวน argument ของโปรแกรม และ argument อื่นๆ
  - ต้องเรียกเป็นคำสั่งแรกก่อนคำสั่งอื่นของ GLUT ทั้งหมด
  - ความจริงไม่มีอะไรมาก ปกติเราเขียน
    int main(int argc, char ** argv)
  - ก็แค่ให้เรียก glutInit(&argc, argv) เป็นคำสั่งแรกใน main ก็พอ

glutInitDisplay

- glutInitDisplay(unsigned int mode)
  - เลือกว่าสีของ pixel ในโปรแกรมของเราจะเป็นแบบใด
    - มีให้เลือกสองแบบคือ RGB กับ Indexed Color
    - เราจะไม่ใช้ Indexed Color เลย
  - เลือกว่าจะใช้ single buffer หรือ double buffer
    - ใช้ double buffer จะทำให้ animation ดูลื่นไหลกว่า
  - เลือกว่าจะให้มี buffer อื่นๆ นอกจาก buffer สีอะไรบ้าง
    - ปกติจะใช้แค่ depth buffer สำหรับเก็บความลึกของจุดแต่ละจุด
  - ค่า mode เกิดจากการเอาค่าคงที่ของตัวเลือกต่างๆ มา or กัน
    - ปกติเราจะใช้ GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH
    - กรณีของ code ตัวอย่างใช้ GLUT_SINGLE | GLUT_RGBA

คำสั่งสำหรับจัดการวินโดวส์

- glutCreateWindow(char *string)
  - สร้างวินโดวที่มี title เป็น string ที่ให้
- glutInitWindowPosition(int x, int y)
  - กำหนดตำแหน่งขอบบนของวินโดว
- glutInitWindowSize(int width, int height)
  - กำหนดขนาดของวินโดว

glutDisplayFunc

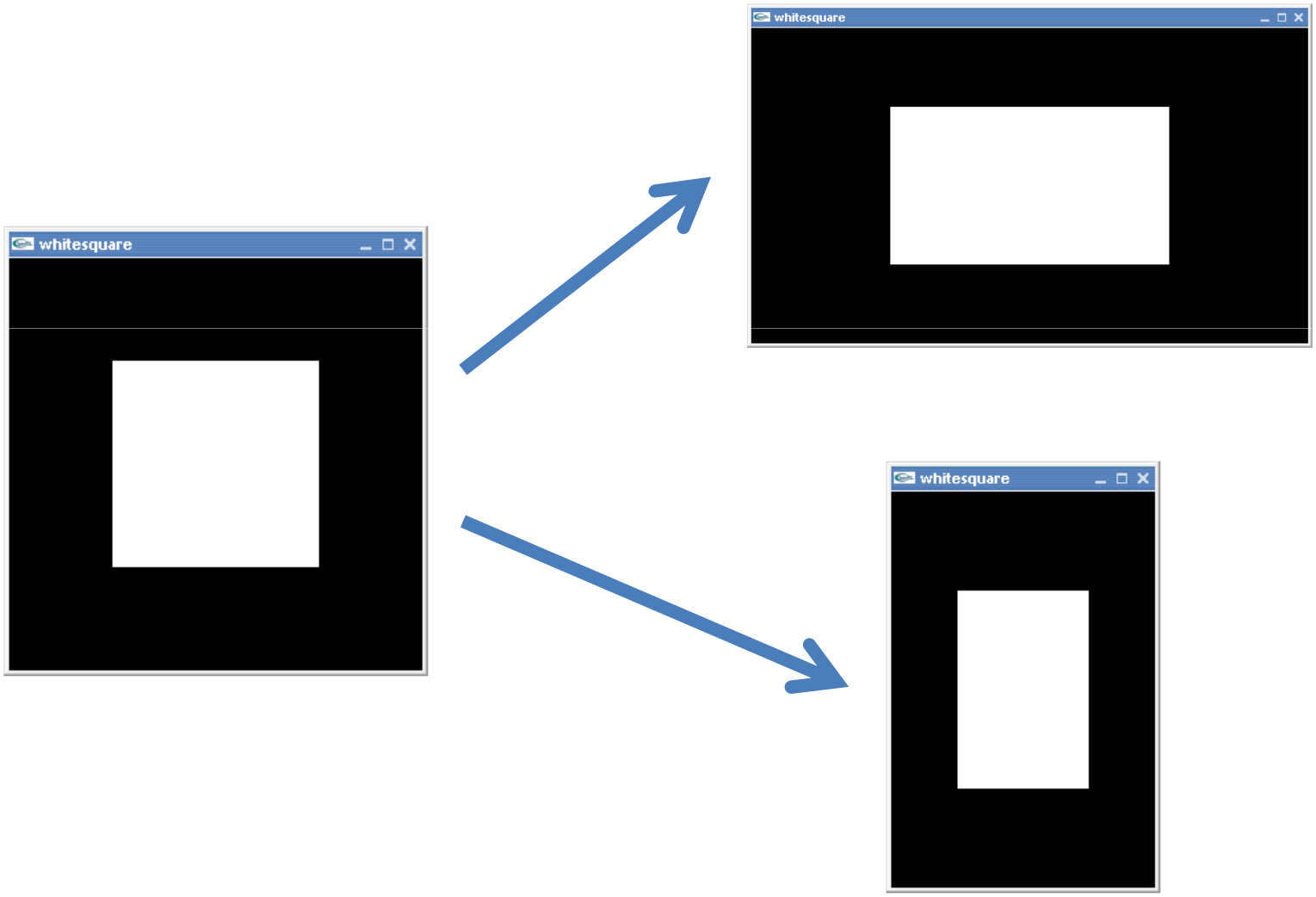
- glutDisplayFunc(void (*func)(void))
  - กำหนดฟังก์ชันที่ GLUT จะเรียกทุกครั้งเมื่อมันต้องวาดหน้าจอใหม่
  - ฟังก์ชันที่จะส่งให้ glutDisplayFunc ต้องมี prototype void <ชื่อฟังก์ชัน>(void)
  - ยกตัวอย่างเช่นฟังก์ชัน void draw() ในโค้ดตัวอย่าง
  - ฟังก์ชันนี้ส่วนมากจะเต็มไปด้วยคำสั่ง OpenGL

glutMainLoop

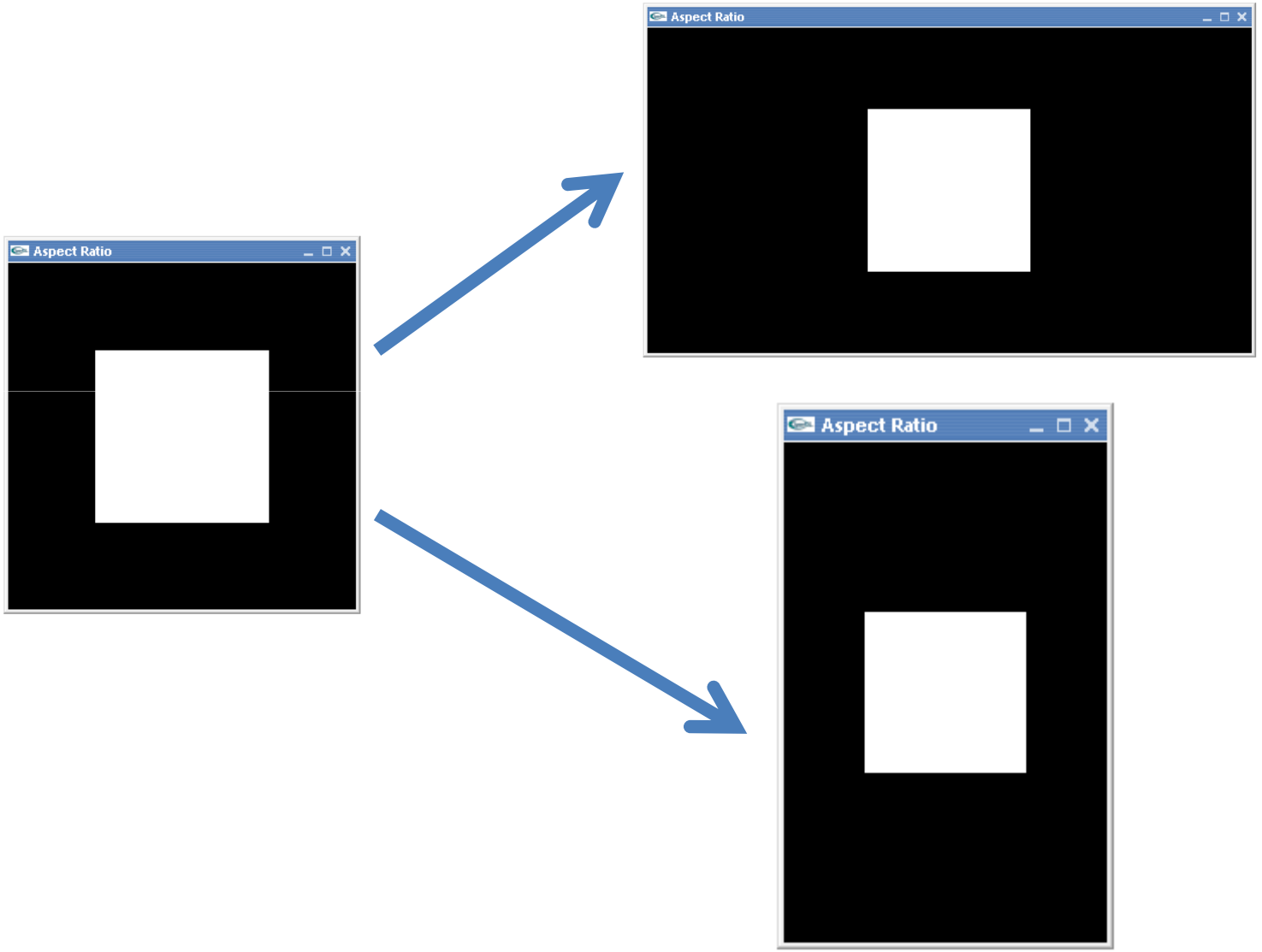
- glutMainLoop()
  - ฟังก์ชันสุดท้ายที่เราเรียกในโปรแกรม
  - สั่งให้ GLUT ไปทำงานของมัน
  - งานของ GLUT
    - รับ input จากผู้ใช้
    - เรียกฟังก์ชันที่ให้ใน glutDisplayFunc
    - เริ่มต้นใหม่อีกครั้ง
  - ระวัง: ต้องสร้าง windows และกำหนด displayFunc ให้เรียบร้อยก่อนเรียก glutMainLoop

การย่อขยายขนาดวินโดว์

เมื่อย่อขยายวินโดว์



ต้องการอย่างนี้มากกว่า



glutReshapeFunc

- glutReshapeFunc(void (*func)(int width, int height))
  - ให้ฟังก์ชันที่รับ int สองตัว
  - ฟังก์ชันที่ให้ไปจะถูกเรียกทุกครั้งที่วินโดว์เปลี่ยนขนาด
  - Argument ที่เป็น int สองตัว
    - ตัวแรกคือความกว้างของหน้าต่างหลังถูกเปลี่ยนความกว้าง หน่วยเป็นพิกเซล
    - ตัวที่สองคือความสูงของหน้าต่างหลังถูกเปลี่ยนความกว้าง หน้วยเป็นพิกเซล
  - เราสามารถใช้ฟังก์ชันที่ให้ glutReshapeFunc ไปเป็นตัวปรับอัตราส่วนของรูปที่แสดงออกมาได้

ตัวอย่างการใช้ glutReshapeFunc

```
void reshape(int w, int h)
{
    .........
}

void draw()
{
    .........
}
```

```
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB
    A | GLUT_SINGLE);
    glutCreateWindow("window");
    glutReshapeFunc(reshape);
    glutDisplayFunc(draw);
    glutMainLoop();
}
```

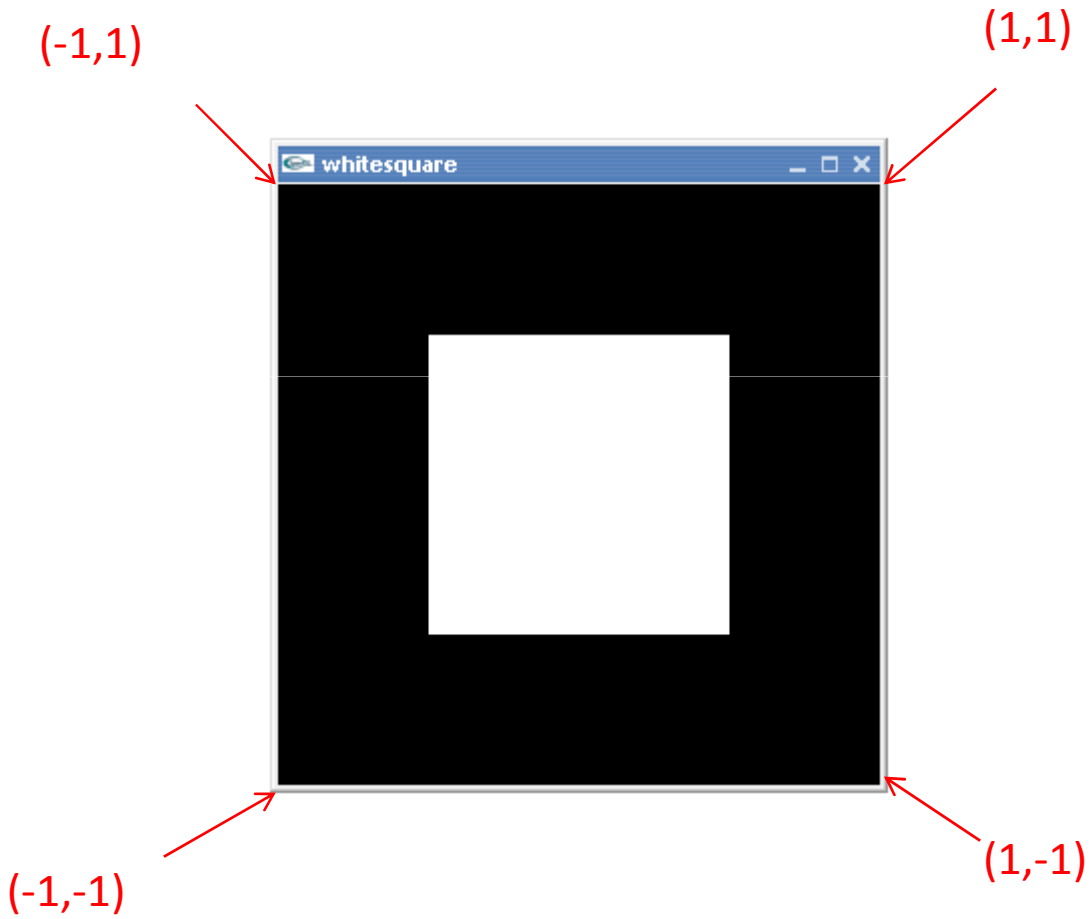
Aspect Ratio

- อัตราส่วนระหว่างความกว้างของรูปต่อความสูงของรูป

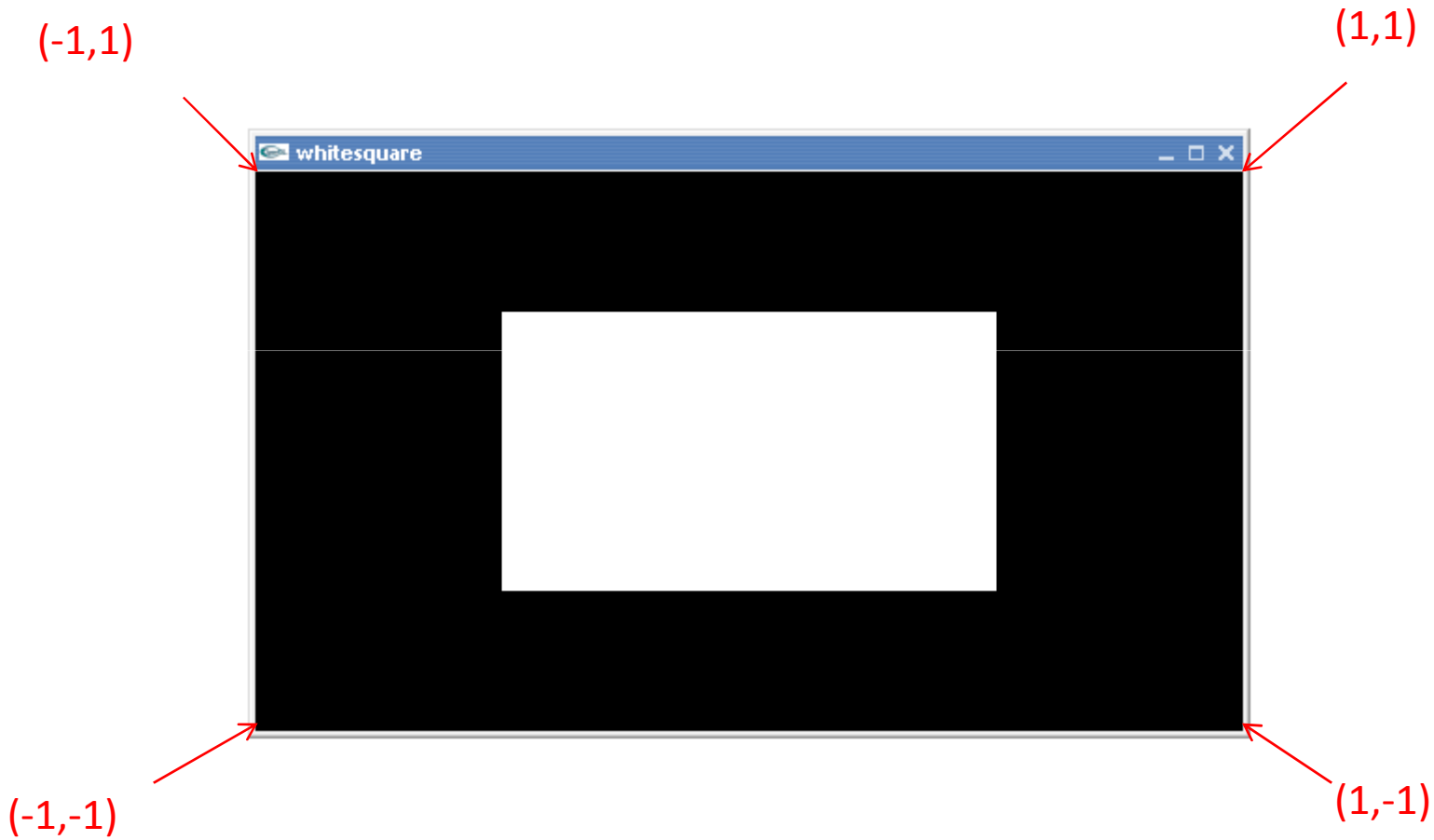
$$\text{aspect ratio} = \frac{\text{width}}{\text{height}}$$

- Aspect ratio ของรูปสี่เหลี่ยมสีขาว = 1
- เวลาย่อขยายหน้าจอ aspect ratio เปลี่ยนแม้เราจะวาดรูปที่จุดเดิมก็ตาม
- นี่เป็นเพราะจุดต่างๆ ย้ายที่

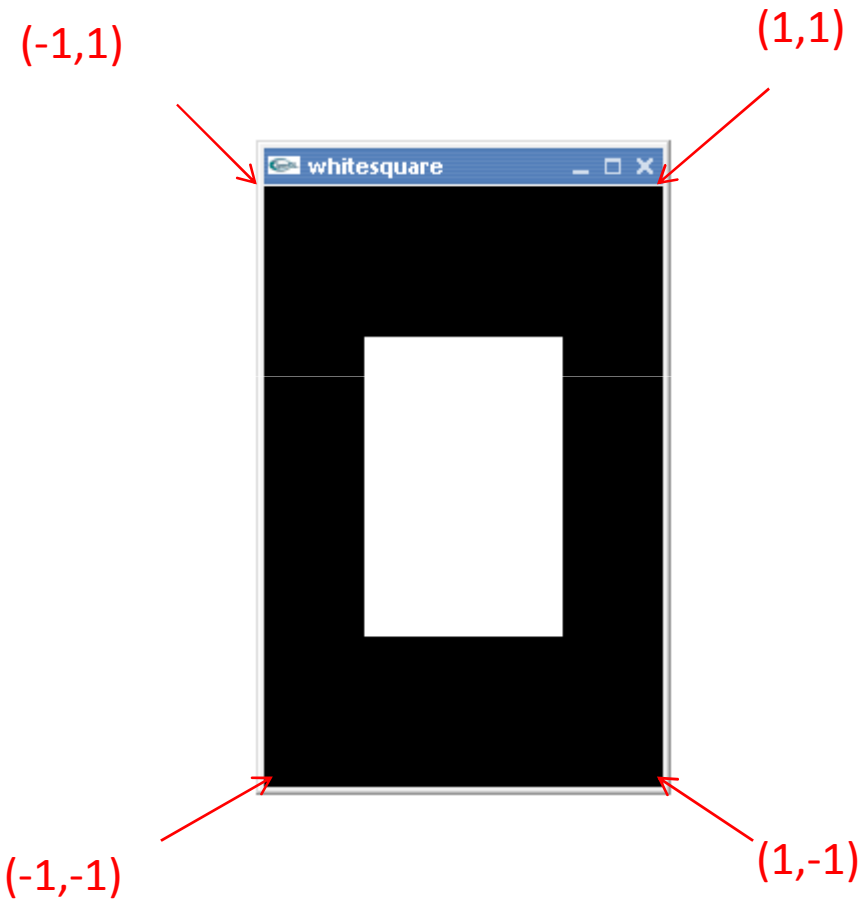
Aspect Ratio (ต่อ)



Aspect Ratio (ต่อ)



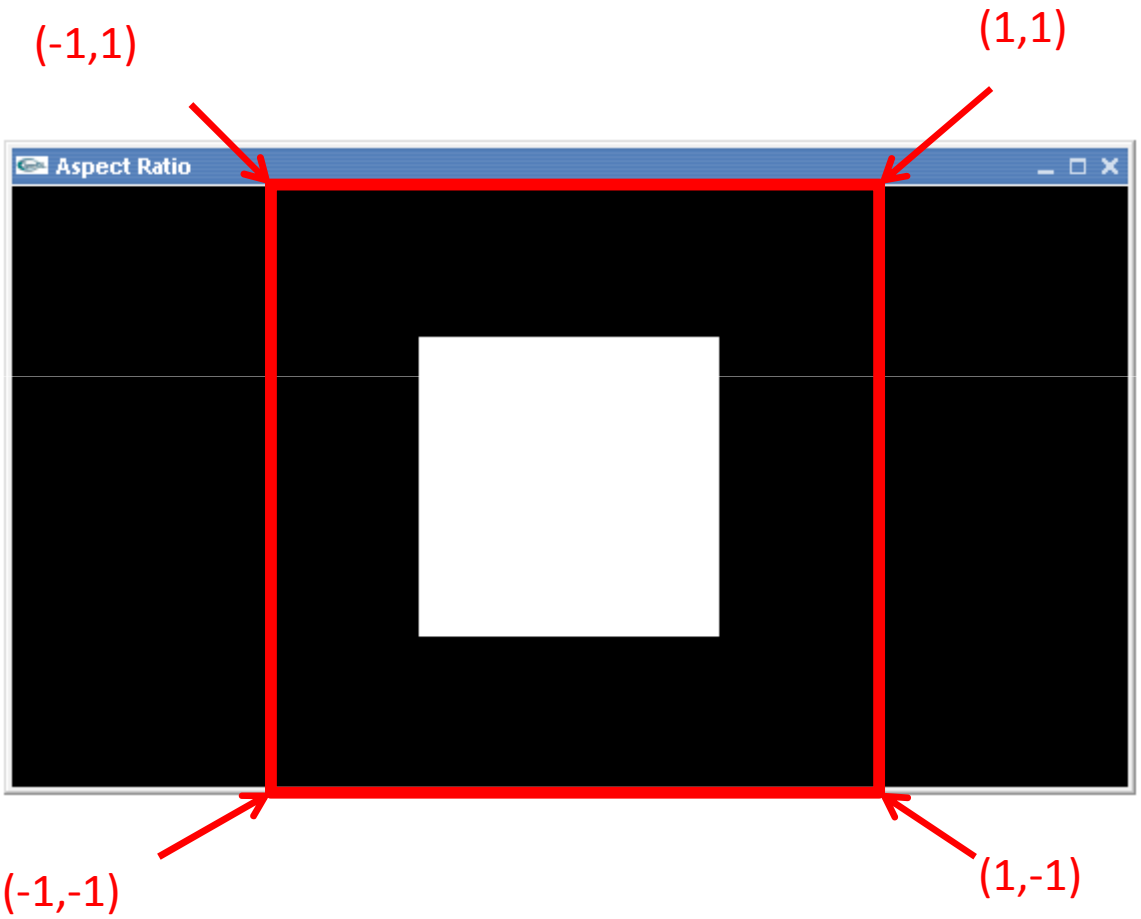
Aspect Ratio (ต่อ)



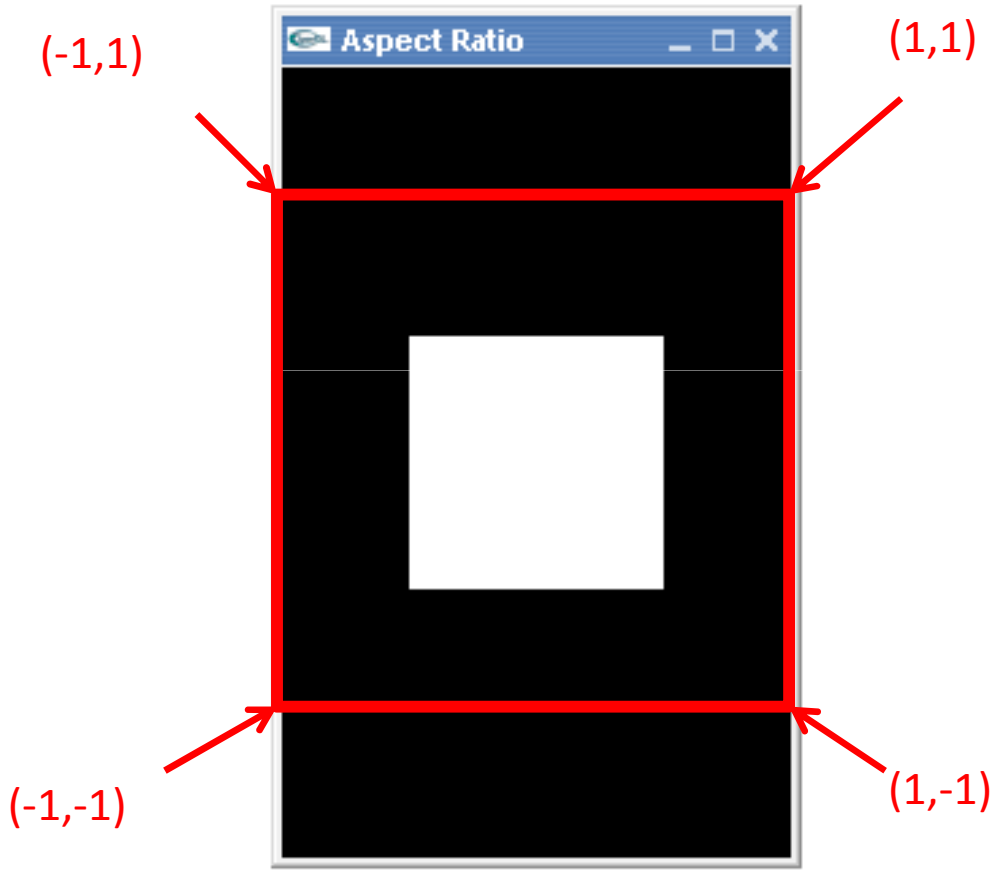
สิ่งที่เราต้องการ

- เวลาย่อหรือขยายวินโดว์ aspect ratio ของรูปควรจะมีค่าคงที่ แม้ว่า aspect ratio ของวินโดว์จะเปลี่ยน
- แต่เราไม่ต้องการวาดรูปใหม่
  - ไม่ต้องการเปลี่ยนฟังก์ชัน draw
- เราสามารถทำได้โดยเปลี่ยนระบบพิกัด (coordinate system) ของพื้นที่ที่เราจะวาดรูปเสียใหม่

กรณีความกว้างมากกว่าความสูง



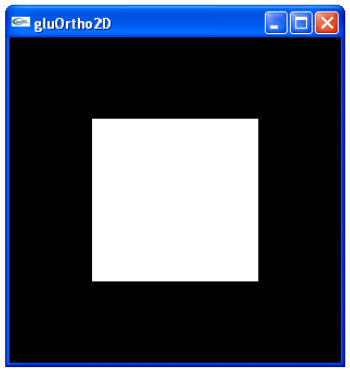
กรณีความสูงมากกว่าความกว้าง



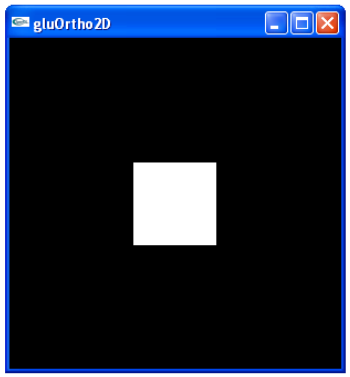
gluOrtho2D

- gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top)
  - ใช้เปลี่ยน projection matrix ให้เป็นการฉายแบบ orthogonal projection
    - เราจะไปพูดถึงคำศัพท์เหล่านี้ในอีกประมาณสองอาทิตย์หน้า
  - ตอนนี้เข้าใจว่าเป็นการเซตพิกัดของจุดมุมของบริเวณที่เราจะวาดรูป
    - มุมล่างซ้ายเป็น (left, bottom)
    - มุมล่างขวาเป็น (right, bottom)
    - มุมบนซ้ายเป็น (left, top)
    - มุมบนขวาเป็น (right, top)

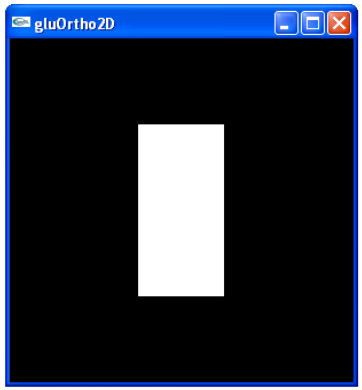
gluOrtho2D (ต่อ)



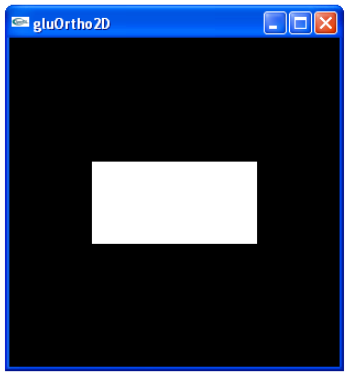
gluOrtho2D(-1,1,-1,1)



gluOrtho2D(-2,2,-2,2)

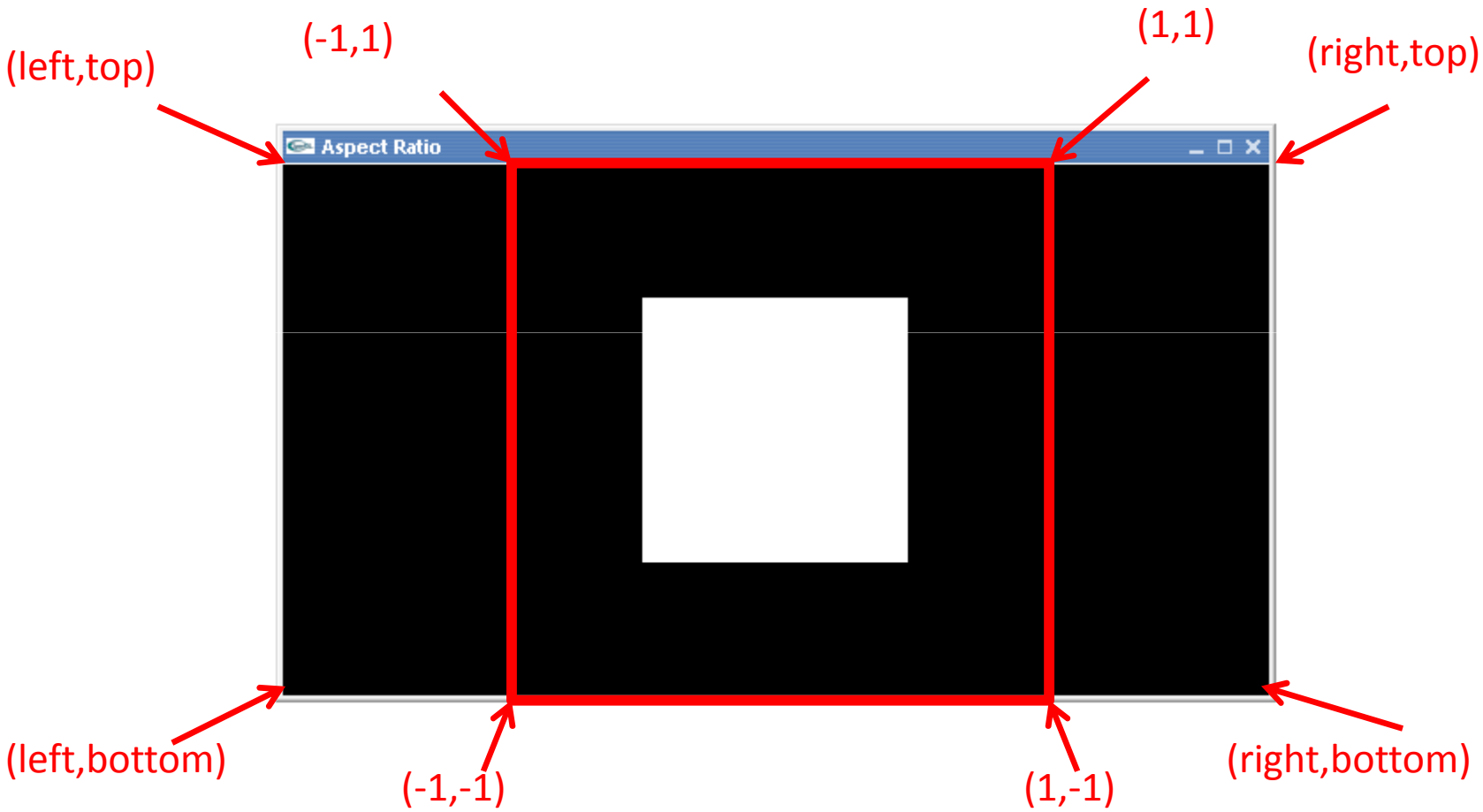


gluOrtho2D(-2,2,-1,1)



gluOrtho2D(-1,1,-2,2)

กรณีความกว้างมากกว่าความสูง (ต่อ)



กรณีความกว้างมากกว่าความสูง (ต่อ)

- bottom = -1
- top = 1
- left = ???, right = ???
- แต่เรารู้ว่า left = -right
- สังเกตว่า

$$\frac{w}{h} = \frac{right - left}{top - bottom}$$

เมื่อ w และ h คือความกว้างและความสูงของวินโดว์ ตามลำดับ

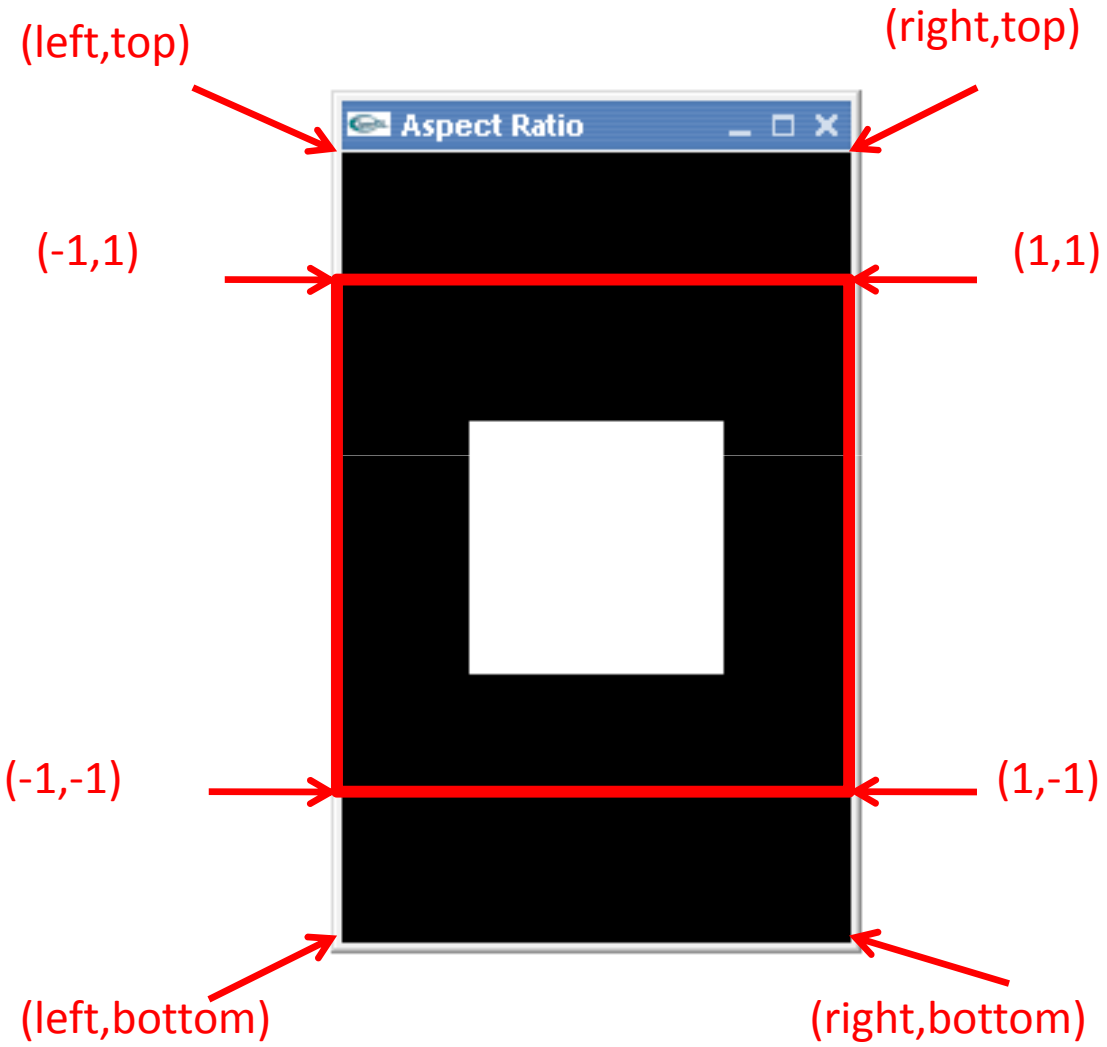
กรณีความกว้างมากกว่าความสูง (ต่อ)

- นั่นคือ

$$\frac{w}{h} = \frac{left + (-left)}{1-(-1)} = \frac{2left}{2} = left$$

- ดังนั้น
  - left = w/h
  - right = -w/h

กรณีความสูงมากกว่าความกว้าง (ต่อ)



กรณีความสูงมากกว่าความกว้าง (ต่อ)

- left = -1
- right = 1
- top = ???, bottom = ???
- เรารู้ว่า bottom = -top

$$\frac{w}{h} = \frac{right - left}{top - bottom} = \frac{1-(-1)}{top-(-top)} = \frac{2}{2top} = \frac{1}{top}$$

- ดังนั้น top = h/w และ bottom = -h/w

Callback สำหรับเวลาวินโดว์เปลี่ยนขนาด

```
void reshape(int w, int h)
{
   glViewport(0,0,w,h);

   if (w == 0) w = 1;
   if (h == 0) h = 1;

   double aspect = w * 1.0 / h;

   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   if (w > h)
       gluOrtho2D(-aspect, aspect, -1, 1);
   else
       gluOrtho2D(-1, 1, -1/aspect, 1/aspect);
}
```

Callback สำหรับเวลาวินโดว์เปลี่ยนขนาด (ต่อ)

```
void reshape(int w, int h)
{
   glViewport(0,0,w,h);

   if (w == 0) w = 1;
   if (h == 0) h = 1;

   double aspect = w * 1.0 / h;

   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   if (w > h)
       gluOrtho2D(-aspect, aspect, -1, 1);
   else
       gluOrtho2D(-1, 1, -1/aspect, 1/aspect);
}
```

กัน division by zero

Callback สำหรับเวลาวินโดว์เปลี่ยนขนาด (ต่อ)

```
void reshape(int w, int h)
{
   glViewport(0,0,w,h);

   if (w == 0) w = 1;
   if (h == 0) h = 1;

   double aspect = w * 1.0 / h;

   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   if (w > h)
       gluOrtho2D(-aspect, aspect, -1, 1);
   else
       gluOrtho2D(-1, 1, -1/aspect, 1/aspect);
}
```

คำนวณ aspect ratio ของวินโดว์

Callback สำหรับเวลาวินโดว์เปลี่ยนขนาด (ต่อ)

```
void reshape(int w, int h)
{
   glViewport(0,0,w,h);

   if (w == 0) w = 1;
   if (h == 0) h = 1;

   double aspect = w * 1.0 / h;

   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   if (w > h)
       gluOrtho2D(-aspect, aspect, -1, 1);
   else
       gluOrtho2D(-1, 1, -1/aspect, 1/aspect);
}
```

กว้าง > สูง

Callback สำหรับเวลาวินโดว์เปลี่ยนขนาด (ต่อ)

```
void reshape(int w, int h)
{
   glViewport(0,0,w,h);

   if (w == 0) w = 1;
   if (h == 0) h = 1;

   double aspect = w * 1.0 / h;

   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   if (w > h)
       gluOrtho2D(-aspect, aspect, -1, 1);
   else
       gluOrtho2D(-1, 1, -1/aspect, 1/aspect);
}
```

สูง > กว้าง

glViewport

- void glViewport(GLint *x*, GLint *y*,
  GLsizei *width*, GLsizei *height*)
  - กำหนดพื้นที่ในวินโดว์ที่จะใช้แสดงผลภาพที่ OpenGL สร้าง
  - x, y, width, height มีหน่วยเป็นพิกเซล
  - พิกัด (x,y) กำหนดตำแหน่งมุมบนซ้ายของพื้นที่
  - width กำหนดความกว้างของพื้นที่
  - height กำหนดความสูงของพื้นที่
- ในตัวอย่างเราใช้ glViewport(0,0,w,h) หมายความว่าเราใช้พื้นที่ทั้งหมดของวินโดว์

Callback สำหรับเวลาวินโดว์เปลี่ยนขนาด (ต่อ)

- เราเรียก

  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();

  ก่อนจะเรียก

  glOrtho2D(…);

  เพื่อกำหนดระบบพิกัด
- ทั้งสองฟังก์ชันนี้เกี่ยวข้องกับการกำหนด viewing transform ซึ่งเราจะพูดถึงในสองสัปดาห์หน้า
- ตอนนี้ให้จำไปก่อนว่าต้องเรียนสองฟังก์ชันนี้ก่อนใช้ glOrtho2D เสมอ

วัตถุเรขาคณิตใน OPENGL

วัตถุเรขาคณิตใน OpenGL

- OpenGL สามารถวาดวัตถุเรขาคณิตง่ายๆ ได้ 3 อย่าง
  - จุด
  - ส่วนของเส้นตรง
  - รูปหลายเหลี่ยม
- ไม่สามารถวาดเส้นโค้งหรือพื้นผิวโค้งได้
- แต่เราสามารถวาดเส้นโค้งด้วยการวาดเส้นตรงสั้นๆ หลายเส้น

Vertex

- การกำหนดวัตถุทางเรขาคณิตใน OpenGL ทำได้โดยการกำหนด vertex หรือ “จุดมุม” ของวัตถุนั้น
  - จุดใน OpenGL มี 1 vertex
  - ส่วนของเส้นตรงใน OpenGL มี 2 vertices (เพราะส่วนของเส้นตรงเกิดจากการลากเส้นเชื่อมจุดสองจุด)
  - รูปหลายเหลี่ยมมีจำนวน vertex เท่ากับจำนวนเหลี่ยม
    - สามเหลี่ยมมี 3 vertices
    - สี่เหลี่ยมมี 4 vertices
    - n เหลี่ยมมี n vertices

glVertex

- glVertex[234][sifd][v](TYPE coords)
  - ใช้กำหนดตำแหน่ง vertex
  - สามารถมี argument 2, 3, หรือ 4 ตัวก็ได้
    - สี่ตัวตรงกับพิกัดแนว x, y, z, w
      - เราจะพูดถึงพิกัดแนว w ในสัปดาห์หน้าเมื่อเรียนเรื่อง homogeneous coordinate
    - ถ้ามี 3 ตัว จะเข้าใจว่า w = 1
    - ถ้ามี 2 ตัว จะเข้าใจว่า z = 0 และ w = 1
    - ปกติจะใช้ argument แค่ 3 ตัว
  - ตัวอย่าง:
    - glVertex2i(10, 5)
    - glVertex3d(8, 7, 3.14153265)

glVertex (ต่อ)

- เติม v ถ้าต้องการให้ argument เป็น pointer ไปยัง array ของพิกัด
  - GLint p0[] = {1,2,3};
    glVertex3iv(p0);
  - GLfloat *p1 = {2.0f, 3.0f, 4.0f, 5.0f};
    glVertex4fv(p1)

การกำหนดวัตถุเรขาคณิต

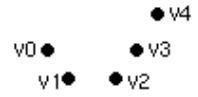
- เริ่มด้วย glBegin(ชนิดของวัตถุ)
- หลังจากนั้นใช้ glVertex กำหนด vertex ของวัตถุนั้น
- แล้วจบด้วย glEnd()
- ที่เคยเห็นมาจากการบรรยายครั้งที่สอง:

```
glBegin(GL_POLYGON);
  glVertex3f(-0.5f, -0.5f, 0.0f);
  glVertex3f(0.5f, -0.5f, 0.0f);
  glVertex3f(0.5f, 0.5f, 0.0f);
  glVertex3f(-0.5f, 0.5f, 0.0f);
glEnd();
```

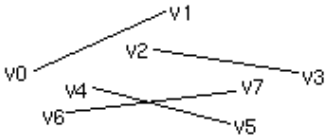
ชนิดของวัตถุ

ค่าที่เอาไปใส่ใน **glBegin(...)**	ชนิดของวัตถุ
GL_POINTS	จุด
GL_LINES	ส่วนของเส้นตรง
GL_LINE_STRIP	ส่วนของเส้นตรงต่อกันหลายเส้น ปลายเปิด
GL_LINE_LOOP	ส่วนของเส้นตรงต่อกันหลายเส้น ปลายปิด
GL_TRIANGLES	สามเหลี่ยม
GL_TRIANGLE_STRIP	สามเหลี่ยมต่อกันเป็นสาย
GL_TRIANGLE_FAN	สามเหลี่ยมต่อกันเป็นรูปพัด
GL_QUADS	สี่เหลี่ยม
GL_QUAD_STRIP	สี่เหลี่ยมต่อกันเป็นสาย
GL_POLYGON	รูปหลายเหลี่ยม

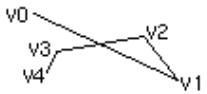
ชนิดของวัตถุ (ต่อ)



GL_POINTS

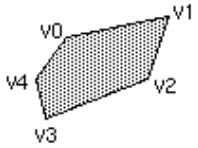


GL_LINES

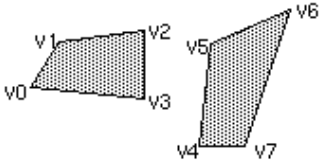


GL_LINE_STRIP

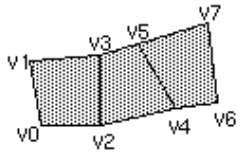
GL_LINE_LOOP



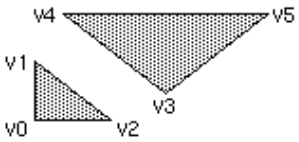
GL_POLYGON



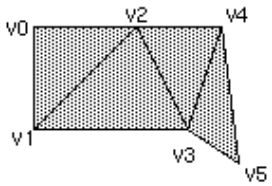
GL_QUADS



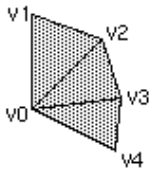
GL_QUAD_STRIP



GL_TRIANGLES



GL_TRIANGLE_STRIP



GL_TRIANGLE_FAN

glColor

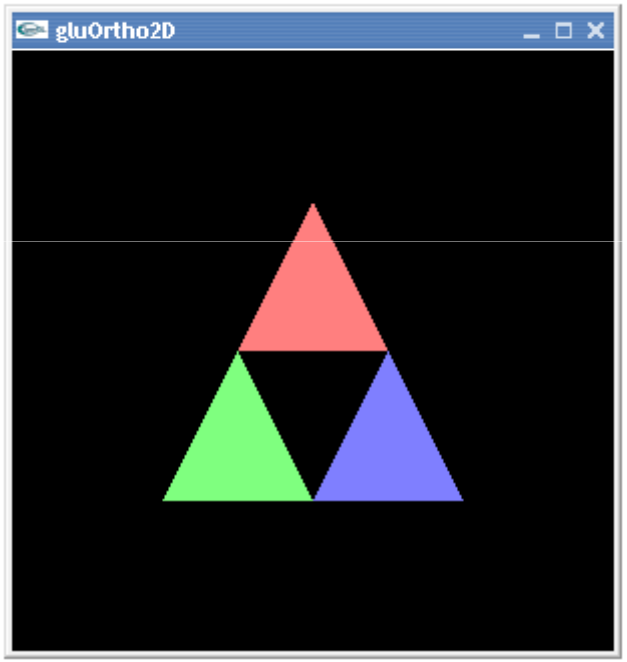
- glColor[34][fd][v](TYPE colors)
  - ใช้กำหนดสีให้กับ vertex
  - กำหนดให้แล้ว vertex จะมีสีนั้นไปจนกว่าจะเรียน glColor เพื่อเปลี่ยนมัน
  - สามารถมี argument 3, หรือ 4 ตัวก็ได้
    - Argument คือ r (สีแดง), g (สีเขียว), b (สีน้ำเงิน), a (ความโปร่งแสง)
    - แต่ละตัวมีค่าตั้งแต่ 0.0 (ไม่มีความเข้มเลย) ถึง 1.0 (เข้มเต็มที่)
    - ถ้ามี argument สามตัว a จะมีค่าเท่ากับ 1.0 (ทึบแสง)
  - ตัวอย่าง
    - glColor3f(1.0f, 1.0f, 0.0f) = สีเหลือง
    - glColor4d(0.5f, 0.5f, 0.5f, 0.5f) = สีเทา โปร่งใส 50%

glColor (ต่อ)

- เติม v ถ้าต้องการให้ argument เป็น pointer ไปยัง array ของสี
  - GLdouble color0[] = {0,1,1};
    gColor3dv(p0);
  - GLfloat *color1 = {0.1f, 0.9f, 0.5f, 0.75f};
    glVert4fv(p1)
- ความโปร่งแสงจะไม่มีผลจนกว่าเราจะบอก OpenGL ให้จัดการความโปร่งแสงให้ (เรื่องนี้เราจะไปพูดก่อนสอบกลางภาคเล็กน้อย)

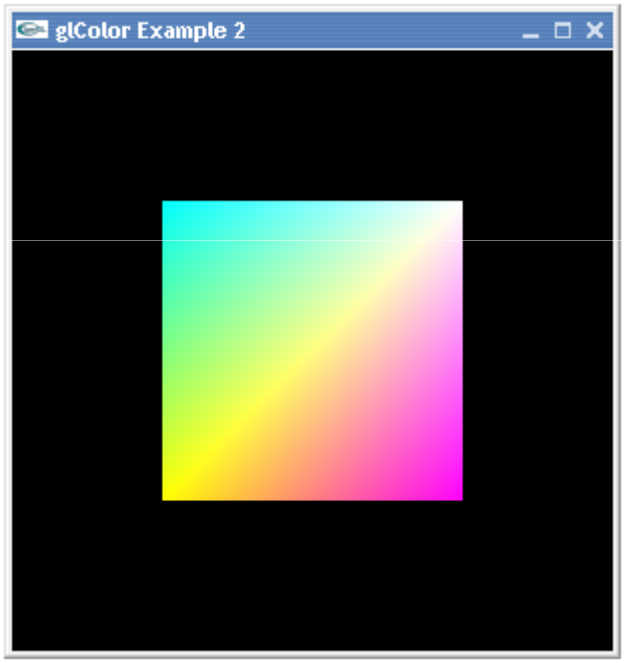
ตัวอย่าง

```
glBegin(GL_TRIANGLES);

    // Red
    glColor3f(1.0f, 0.5f, 0.5f);
    glVertex3f( 0.0f,  0.5f, 0.0f);
    glVertex3f(-0.25f, 0.0f, 0.0f);
    glVertex3f( 0.25f, 0.0f, 0.0f);

    // Green
    glColor3f(0.5f, 1.0f, 0.5f);
    glVertex3f( -0.25f,  0.0f, 0.0f);
    glVertex3f( -0.5f,  -0.5f, 0.0f);
    glVertex3f(  0.0f,  -0.5f, 0.0f);

    // Blue
    glColor3f(0.5f, 0.5f, 1.0f);
    glVertex3f( 0.25f,  0.0f, 0.0f);
    glVertex3f( 0.0f,  -0.5f, 0.0f);
    glVertex3f( 0.5f,  -0.5f, 0.0f);

glEnd();
```



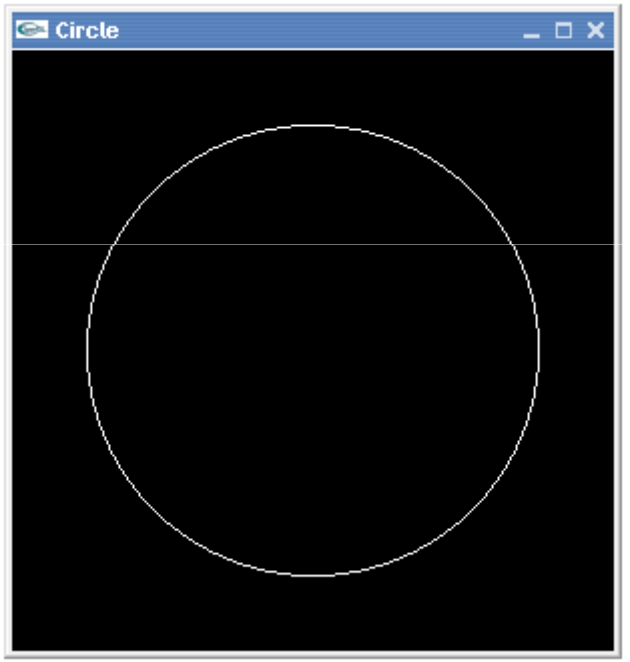
ตัวอย่าง

```
glBegin(GL_QUADS);

    glColor3f(  1.0f,  1.0f, 0.0f);
    glVertex3f(-0.5f, -0.5f, 0.0f);

    glColor3f(  1.0f,  0.0f, 1.0f);
    glVertex3f( 0.5f, -0.5f, 0.0f);

    glColor3f(  1.0f,  1.0f, 1.0f);
    glVertex3f( 0.5f,  0.5f, 0.0f);

    glColor3f(  0.0f,  1.0f, 1.0f);
    glVertex3f(-0.5f,  0.5f, 0.0f);

glEnd();
```



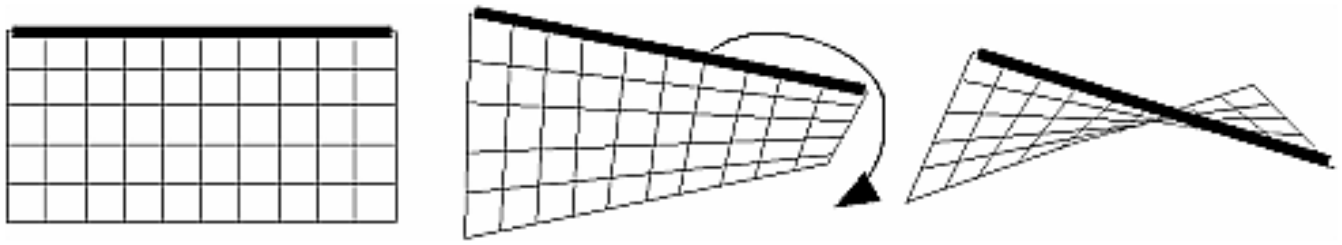
ตัวอย่าง

```
glBegin(GL_LINE_LOOP);
for(int i=0;i<256;i++)
{
   double theta = 2*i*M_PI/256;
   double y = 0.75*sin(theta);
   double x = 0.75*cos(theta);
   glVertex2d(x,y);
}
glEnd();
```



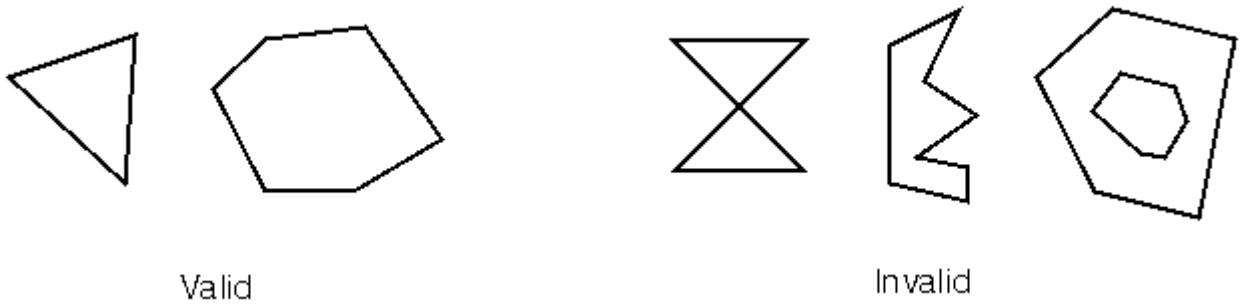
เกี่ยวกับรูปหลายเหลี่ยม

- OpenGL รับประกันว่าจะวาดรูปหลายเหลี่ยมที่จุดทั้งหมดอยู่ในระนาบเดียวกันได้ถูกต้อง
  - ถ้าไม่เป็นเช่นนั้นจะไม่รับประกันว่าจะถูกต้องหรือไม่
- **ข้อสังเกต:** จุดทุกจุดที่อยู่บนรูปสามเหลี่ยมอยู่บนระนาบเดียวกัน
  - แต่นี่ไม่เป็นจริงสำหรับสี่เหลี่ยมหรือรูปหลายเหลี่ยมอื่น



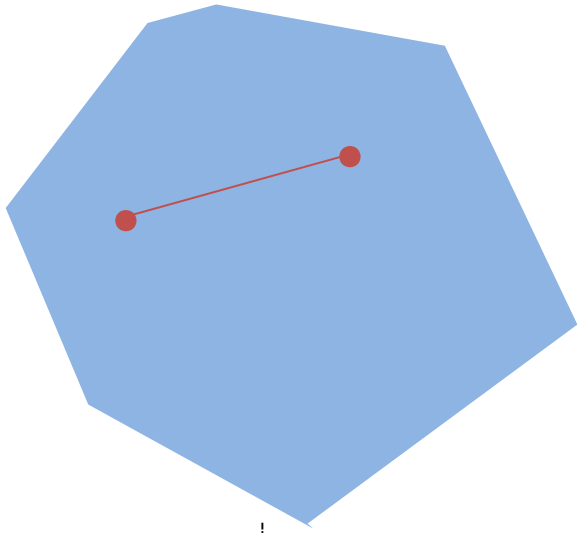
เกี่ยวกับรูปหลายเหลี่ยม (ต่อ)

- รูปหลายเหลี่ยมที่วาดได้ด้วย glBegin(GL_POLYGON) จะมีสมบัติดังนี้
  - เส้นขอบของมันจะต้องไม่ตัดกัน
  - รูปหลายเหลี่ยมนั้นจะต้องเป็นรูปหลายเหลี่ยมนูน
  - รูปหลายเหลี่ยมนั้นจะต้องไม่มี “รู”

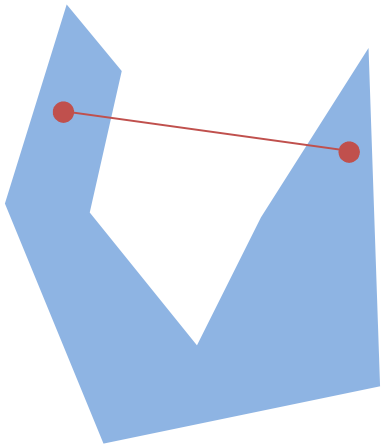


รูปหลายเหลี่ยมนูน (convex polygon)

- สำหรับจุดสองจุดใดๆ ที่อยู่ในรูปหลายเหลี่ยม เมื่อลากส่วนของเส้นตรงเชื่อมจุดสองจุดนั้น ส่วนของเส้นตรงนั้นต้องอยู่ในรูปหลายเหลี่ยมนั้นด้วย
- **ข้อสังเกต:** สามเหลี่ยมเป็นรูปหลายเหลี่ยมนูนเสมอ



รูปหลายเหลี่ยมนูน



ไม่ใช่รูปหลายเหลี่ยมนูน

รูปหลายเหลี่ยมใดๆ

- แล้วเราจะวาดรูปหลายเหลี่ยมที่ไม่ใช่รูปหลายเหลี่ยมนูน หรือรูปหลายเหลี่ยมที่มีรูอย่างไร?
- แยกรูปหลายเหลี่ยมเหล่านั้นออกเป็นรูปหลายเหลี่ยมนูนหลายๆ รูป