

CSE4170 기초 컴퓨터 그래픽스

(2019년 3월)

**서강대학교 공과대학 컴퓨터공학과
임 인 성 교수**

[CSE4170 기초 컴퓨터 그래픽스]

2019년도 1학기

강의자료 I

Introduction to CSE4170

과목 소개

- **과목명:** 기초 컴퓨터 그래픽스 (Introduction to Computer Graphics)
- **과목 번호 및 학점:** CSE4170 / 3학점
- **강의실 및 강의 시간:** K304, 화목 12:00-13:15
- **담당 교수:** 임 인 성 (AS-905, ihm@sogang.ac.kr,
<http://grmanet.sogang.ac.kr/~ihm>)
- **담당 조교:** 주 예 슬(AS-907, jooyesle@sogang.ac.kr)
- **과목 홈페이지:** <http://grmanet.sogang.ac.kr> 참조
 - 조교가 수시로 중요 공지 사항을 본 과목 홈페이지에 게시할 예정임.
 - 본인의 책임하에 수시로 확인할 것.
- **수강대상**
 - 3D 컴퓨터 그래픽스 기술에 관심이 있는 학부 3-4학년생
 - 컴퓨터공학과 3학년 1학기생 수준의 C/C++ 프로그래밍 능력
 - 공대생 2학년 수준의 삼각함수/벡터/행렬/기하에 대한 지식

공학인증 선수과목 이수 요건은
본인의 책임 하에 확인할 것

교과 목표

- 본 과목에서는 **3D 컴퓨터 그래픽스 분야의 이론 및 실제 기술 적용 방법**을 이해하고, 이를 효과적으로 활용할 수 있는 프로그래밍 기법을 습득함을 목표로 한다.
- 특히 현재 산업체의 수요를 반영하여 실시간 3D 그래픽스 기술의 이해 및 적용 능력, 그 중 **PC 환경 및 모바일 장비 환경에서의 실시간 렌더링 SW의 제작 능력** 습득에 주안점을 두고 강의를 진행한다.
- 또한 이를 통하여 현재 급속도로 성능이 향상되고 있는 **PC 및 모바일 플랫폼 상에서의 GPU computing architecture, 즉 SIMD 기반의 many-core processing 구조**에 대한 이해를 높혀, 향후 CUDA/OpenCL/OpenGL Shader 등을 이용한 massively parallel computing에 대한 기초를 쌓는다.
- 이를 위하여 다음과 같은 그래픽스 프로그래밍 API를 통한 실습을 통하여 산업체에서 요구하는 3D 그래픽스 프로그래밍 능력을 습득한다.
 - **PC 플랫폼 상에서의 OpenGL API를 통한 그래픽스 프로그래밍** (학과 실습실 또는 개인 PC 사용)
 - **Unity 환경에서의 HLSL/Cg 기반 쉐이더 프로그래밍을 통한 모바일 플랫폼용 3D 그래픽스 기술 적용 앱 제작** (학과 실습실 또는 개인 PC 사용/Public-domain Unity 3D SW 사용)
 - ~~**안드로이드 환경의 휴대폰/탭 플랫폼 상에서의 OpenGL ES API를 통한 그래픽스 프로그래밍** (학과 보유 삼성 갤럭시 탭 대여 또는 개인 휴대폰/탭 사용)~~

예상 일정 (2019학년도 1학기)

- 수업 내용, 퀴즈, 프로그래밍 숙제는 수강생들의 수업 이해도에 따라 적절히 조정할 예정임.
- 정규 수업 외에 수업 진도에 따라 저녁 시간에 적절히 조교의 코딩 관련 튜토리얼 시간을 가질 수 있음. 출석 체크는 하지 않으며 본인의 필요에 따라 참석.

Week 1 (3월 4일 개강) 3/5, 3/7 - Intro. to computer graphics - Raster graphics fundamentals	Week 2 3/12, 3/14(개강미사)* - GLUT programming - Geometric transform	Week 3 3/19, 3/21 - Geometric transform - 3D viewing pipeline	Week 4 3/26, 3/28 - OpenGL viewing pipeline
Week 5 4/2, 4/4 - OpenGL viewing pipeline - Lighting models	Week 6 4/9, 4/11 - Lighting models - Lighting programming using OpenGL	Week 7 4/16, 4/18(부활절휴가?) - Lighting programming using OpenGL - Rasterization and interpolation	Week 8 MIDTERM (4/22~)
Week 9 4/30, 5/2 - Texture mapping techniques	Week 10 5/7, 5/9 - OpenGL rendering pipeline review	Week 11 5/14, 5/16 - Intro. To Unity and HLSL/Cg shader programming	Week 12 5/21, 5/23 - Advanced shader programming I
Week 13 5/28, 5/30 - Advanced shader programming II	Week 14 6/4, 6/6(현충일) - Data structured for graphics - Ray tracing method	Week 15 6/11, 6/13, 6/18(보강일) - Ray tracing method - VR/AR/MR/shader programing	Week 16 FINAL (6/19~)

참고 자료

<본 과목 제공 자료>

- 임인성, OpenGL을 통한 3차원 그래픽스 프로그래밍: 기초편, 그린 출판사, 2001. (pdf 파일 형태로 제공 예정)
- 본 과목 강의 자료
- 컴퓨터 그래픽스 연구실 보유 각종 예제 프로그램 (PC용 및 모바일 기기용)

<3D 컴퓨터 그래픽스 전반> (또는 이에 상응하는 교재를 자신이 선택)

- J. Hughes et al., Computer Graphics: Principles and Practice(3rd ed.), Addison-Wesley, 2013.
- S. Marschner et al., Fundamentals of Computer Graphics(4th ed.), CRC Press, 2015.
- E. Angel, Interactive Computer Graphics: A Top-Down Approach with Shader-Based OpenGL (7th ed.), Addison-Wesley, 2014.
- T. Akenine-Möller et al., Real-Time Rendering(4th ed.), AK Peters/CRC Press, 2018.
- 기타

<Open/OpenGL ES 프로그래밍 관련>

- D. Shreiner et al., OpenGL Programming Guide(9th ed.): The Official Guide to Learning OpenGL, Versions 4.5 with SPIR-V, 2016.
- G. Sellers and R. Wright Jr., OpenGL Superbible: Comprehensive Tutorial and reference(7th ed.), Addison-Wesley Professional, 2015.
- D. Wolff, OpenGL 4 Shading Language Cookbook(3nd ed.), Packt Publishing, 2018.
- D. Ginsburg et al., OpenGL ES 3.0 Programming Guide(2nd ed.), Addison-Wesley, 2014.
- 기타 OpenGL 및 OpenGL ES API 관련 기술자료 문서, (<http://www.khronos.org/> 참조)
 - OpenGL 및 OpenGL ES Specifications.
 - OpenGL 및 OpenGL ES Shading Language Specifications.
 - OpenGL 및 OpenGL ES Programming Guides/Reference Manuals.
 - OpenGL 및 OpenGL ES Shading Language Programming Guides/Reference Manuals.

무엇을 배울 것인가?

- 목표
 - Computer graphics의 geometric modeling, animation, rendering 등의 핵심 주제 중 **real-time rendering**에 초점을 맞춤.
 - 산업체에서 가장 우선적으로 필요로 하는 기본 기술
 - 3D 그래픽스 프로그래밍을 통한 문제 해결 능력을 습득하기 위하여 **OpenGL/Unity Shader/ OpenGL-ES** 프로그래밍을 중요하게 다룰 예정임.
 - 가장 중요한 목표는 **3D real-time rendering pipeline**에 대하여 이해를 하는 것임.

The Need for New Generation GPU APIs



OpenGL has evolved over 25 years - API complexity can obscure optimal performance path and hinder portability



GPUs are increasingly compute AND graphics capable + platforms are becoming unified and multi-core



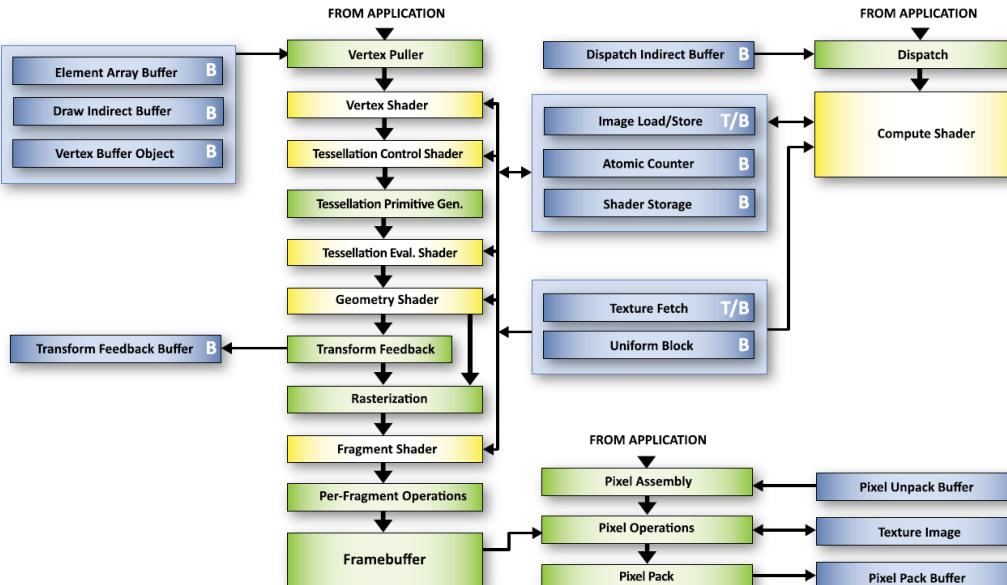
GPUs accelerate graphics, compute, vision and deep learning across diverse platforms:
PORTABILITY is key

OpenGL Pipeline

A typical program that uses OpenGL begins with calls to open a window into the framebuffer into which the program will draw. Calls are made to allocate a GL context which is then associated with the window, then OpenGL commands can be issued.

The heavy black arrows in this illustration show the OpenGL pipeline and indicate data flow.

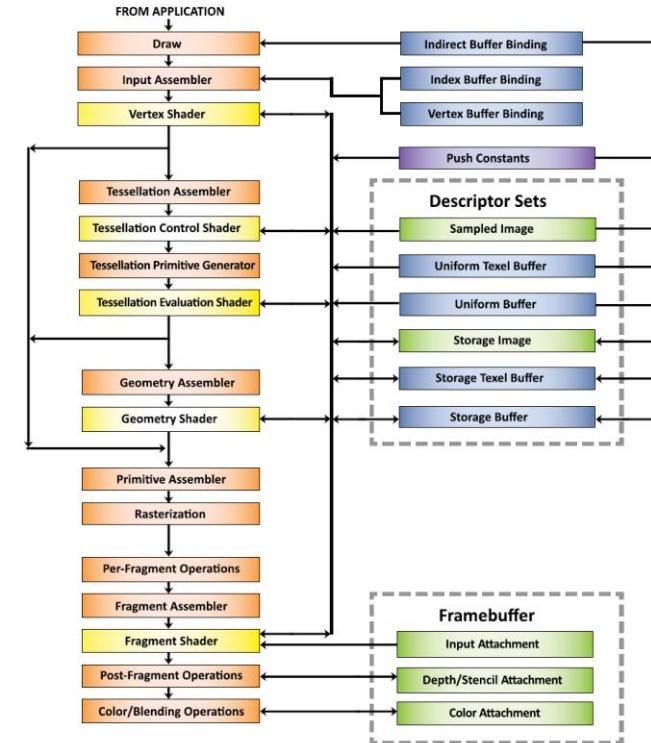
- █ Blue blocks indicate various buffers that feed or get fed by the OpenGL pipeline.
- █ Green blocks indicate fixed function stages.
- █ Yellow blocks indicate programmable stages.
- T Texture binding
- B Buffer binding



OpenGL

Vulkan

Vulkan Pipeline Diagram [9]



Some Vulkan commands specify geometric objects to be drawn or computational work to be performed, while others specify state controlling how objects are handled by the various pipeline stages, or control data transfer between memory organized as images and buffers. Commands are effectively sent through a processing pipeline, either a graphics pipeline or a compute pipeline.

The heavy black arrows in this illustration show the Vulkan graphics and compute pipelines and indicate data flow.

- █ Fixed function stage
- █ Programmable stage
- █ Buffer
- █ Image
- █ Constants

Introduction to 3D Computer Graphics

3D 컴퓨터 그래픽스 기술 응용 분야

- 적용 분야



Special Effects & Animation



3D Games



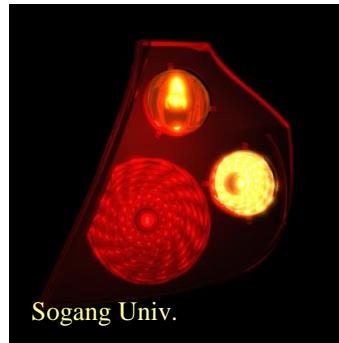
Mobile Graphics



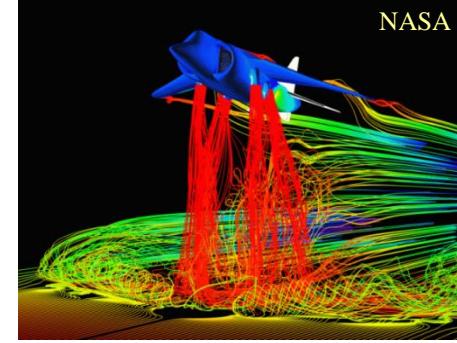
3D User Interfaces



Mixed Reality



Sogang Univ.

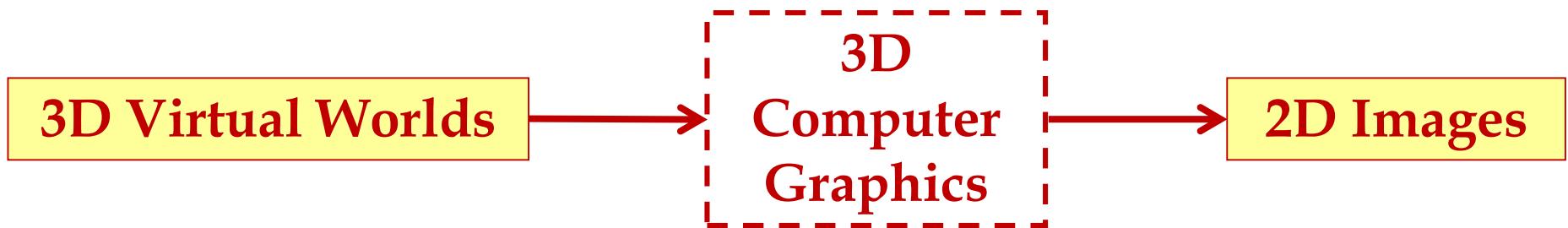


Scientific Visualization
Production
Rendering

3D Computer Graphics

- 목적

- 현실과 같은 가상의 3차원 세상을 창조하여 그로부터 사실적인 영상을 생성하는데 필요한 다양한 기술을 연구

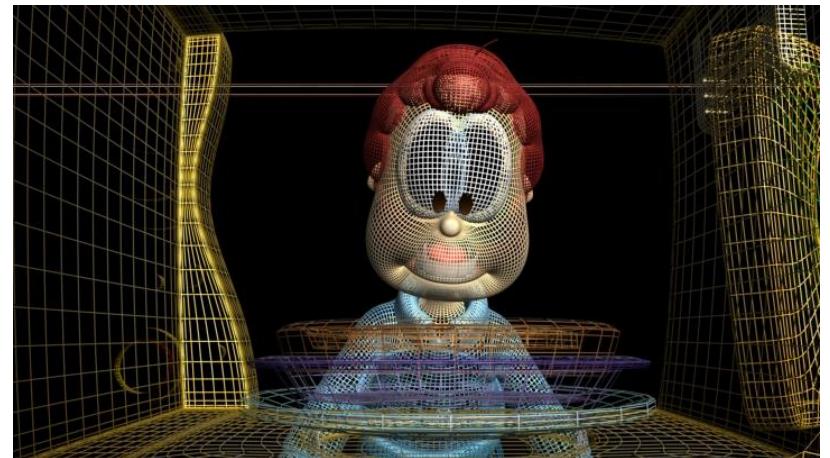
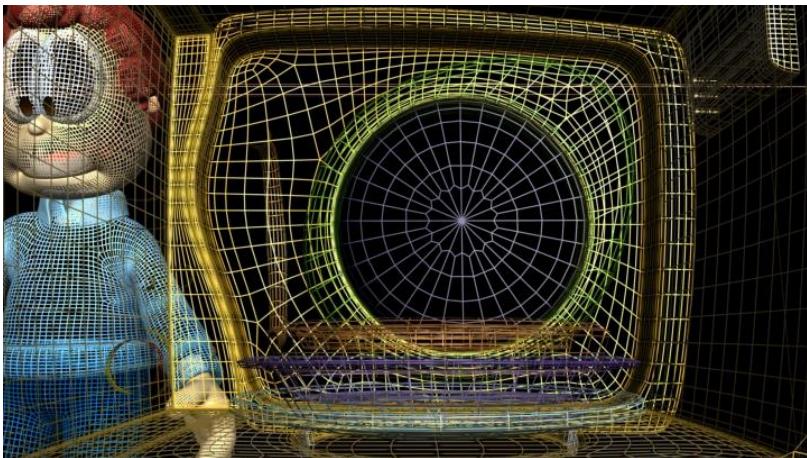


- 전통적인 세 가지 핵심 주제

- 3D 기하 모델링(3D Geometric Modeling)
 - 어떻게 하면 가상의 3차원 세상의 물체들을 효과적으로 표현할 수 있을까?
- 3D 애니메이션(3D Animation)
 - 어떻게 하면 가상의 3차원 세상에서의 움직임을 자연스럽게 표현할 수 있을까?
- 3D 렌더링(3D Rendering)
 - 어떻게 하면 마치 카메라로 촬영한 것과 같이 사실적인 이미지를 생성할 수 있을까?

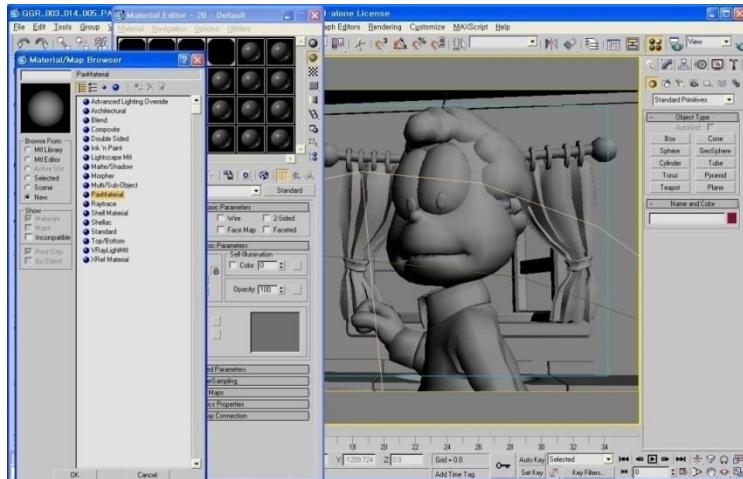
3D Geometric Modeling

- “**3D modeling** is the process of developing a mathematical, wireframe representation of any three-dimensional object, called a "3D model", via specialized software.” (*from Wikipedia*)
 - **Polygonal models**
 - Curved surface models
 - Volumetric models
 - Procedural models
 - Etc.



© 디지아트 프로덕션 & 서강대학교 컴퓨터 그래픽스 연구실

3D modeling tool



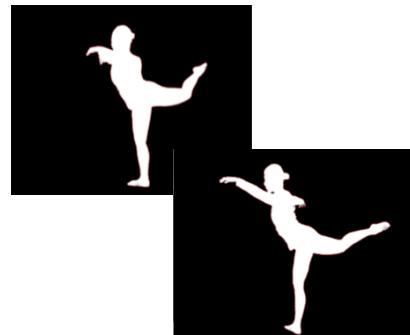
© 디지아트 프로덕션

Scanning

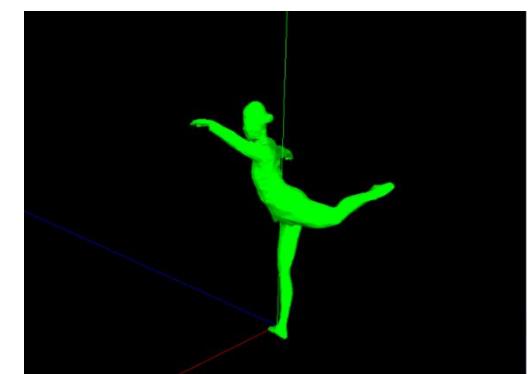


© Pixar

3D models from captured images



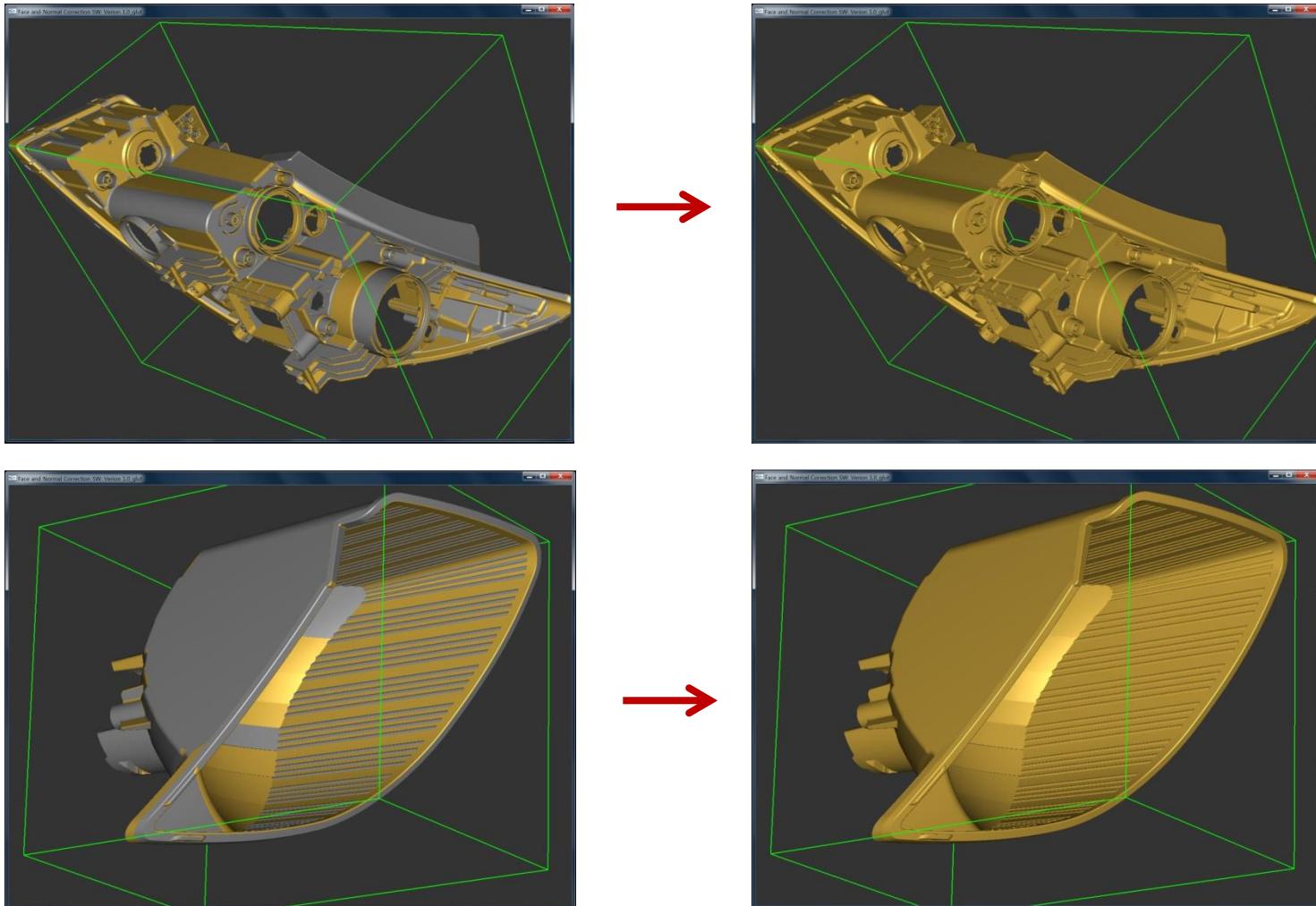
...



© 서강대학교 컴퓨터 그래픽스 연구실 & ETRI

2019년도 1학기 강의자료 I - 15

Geometry Computing Example



© 서강대학교 컴퓨터 그래픽스 연구실 & (주)에스엘

3D Animation

- “**3D animation** refers to the *temporal* description of an object, i.e., how it moves and deforms over time.” (*from Wikipedia*)
 - Key framing
 - Inverse kinematics
 - Motion capture
 - Physically based simulation
 - Etc.

- **Example 1:** Physically based fluid animation



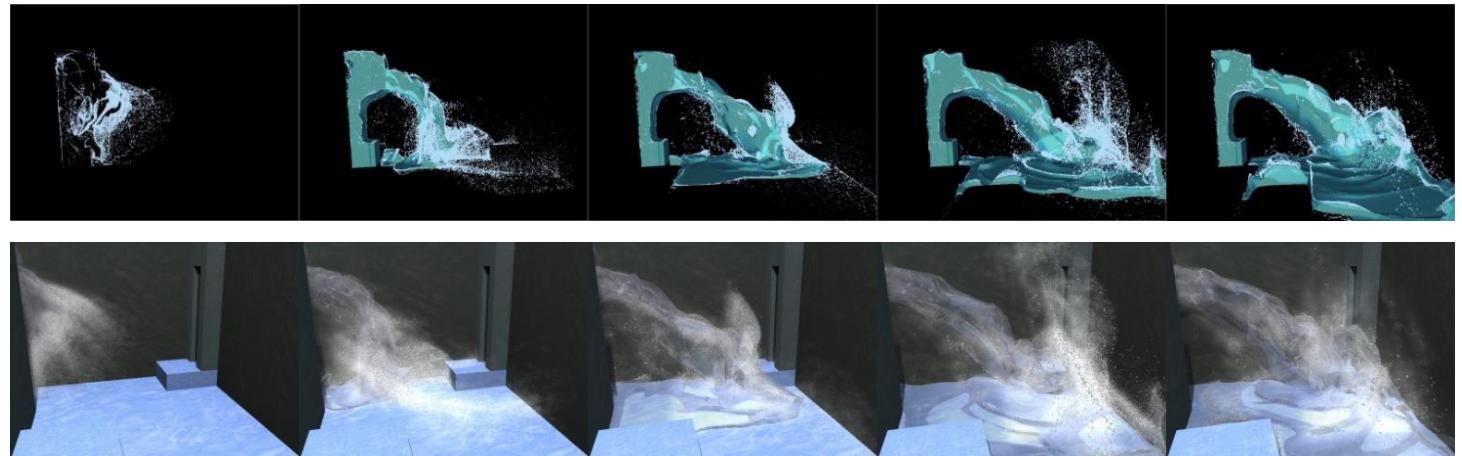
$$\nabla \cdot \mathbf{u} = 0$$

© Nick Foster

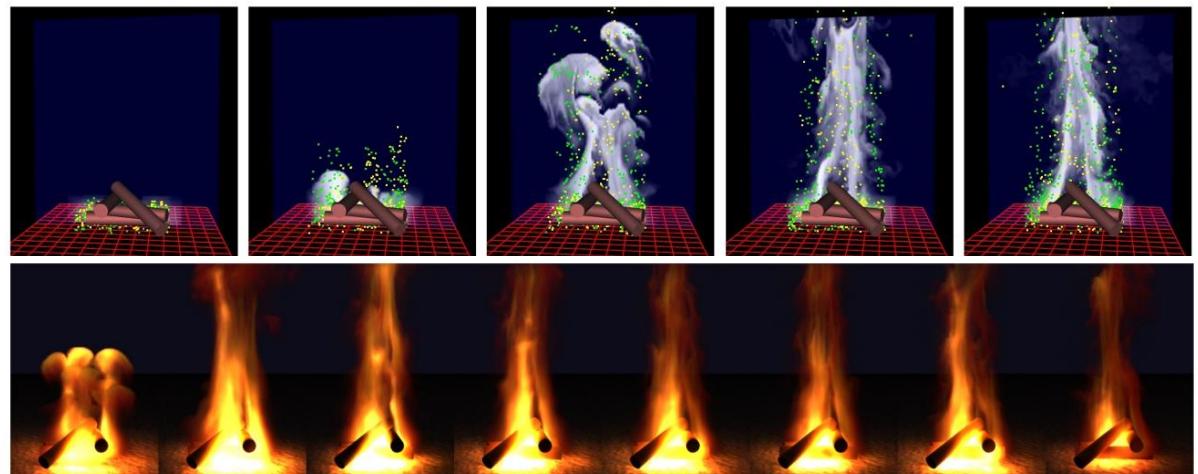
$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla) \mathbf{u} + \nu \nabla \cdot (\nabla \mathbf{u}) - \frac{1}{\rho} \nabla p + \mathbf{f}$$

- **Example 2: Physically based fluid animation**

J. Kim, D. Cha, B. Chang, B. Koo, I. Ihm, "Practical Animation of Turbulent Splashing Water", *ACM SIGGRAPH/Eurographics Symp. on Computer Animation*, 2006.



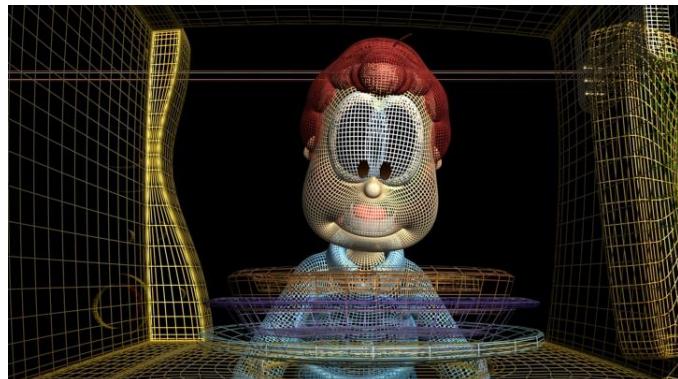
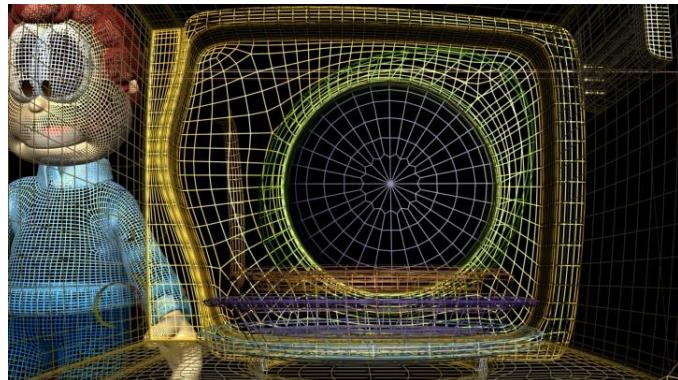
B. Kang, Y. Jang, and I. Ihm, "Animation of Chemically Reactive Fluids Using a Hybrid Simulation Method", *ACM SIGGRAPH/Eurographics Symp. on Computer Animation*, 2007.



© 서강대학교 컴퓨터 그래픽스 연구실

3D Rendering

- “**3D rendering** is the final process of creating the actual 2D image or animation from the prepared scene. This can be compared to taking a photo or filming the scene after the setup is finished in real life.” (*from Wikipedia*)

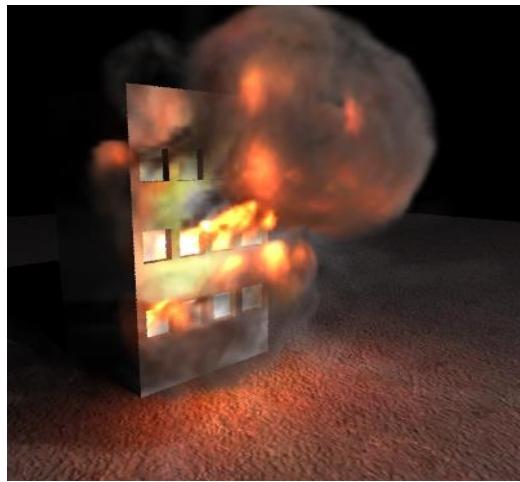
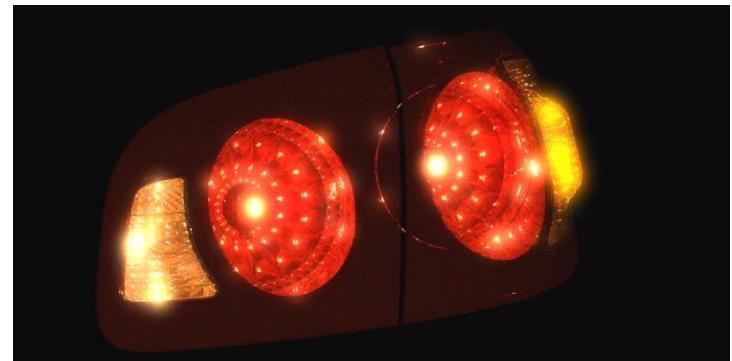
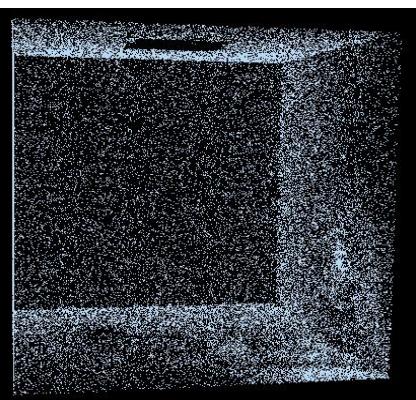
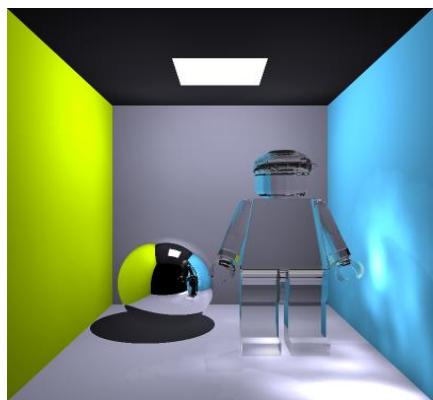


- **Example 1:** Advanced rendering system



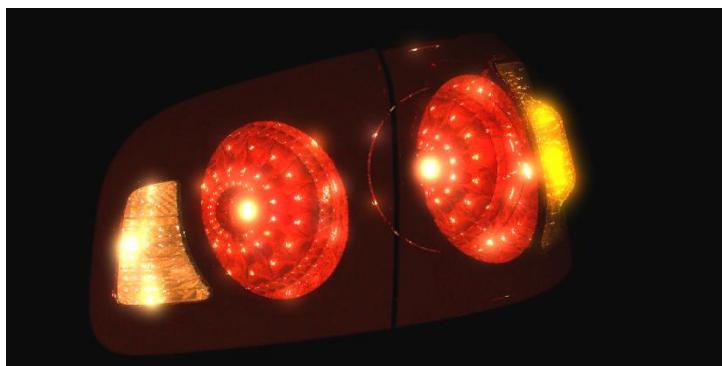
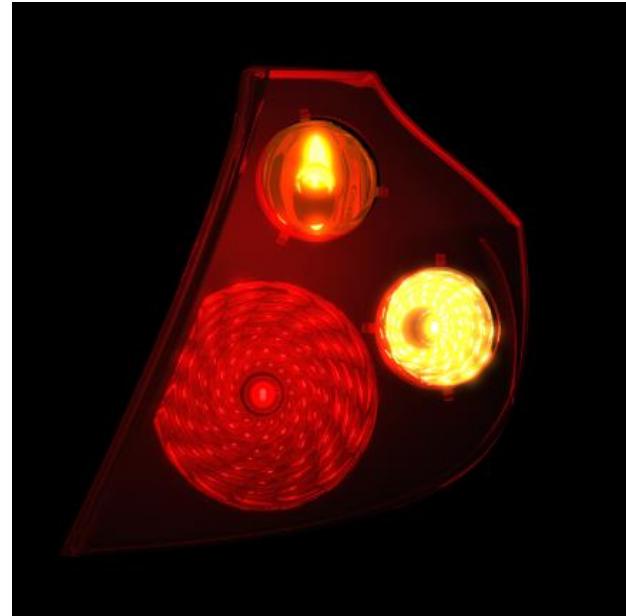
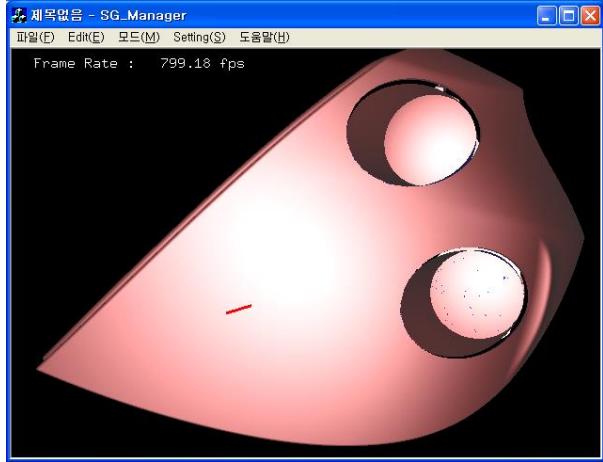
© 서강대학교 컴퓨터 그래픽스 연구실

- **Example 2: Photon-mapping based Monte Carlo ray tracing**



© 서강대학교 컴퓨터 그래픽스 연구실

- **Example 3:** Production rendering



© 서강대학교 컴퓨터 그래픽스 연구실

Photorealistic Rendering

- 마치 카메라로 촬영한 정도 수준의 이미지를 생성하기 위한 기법.
- 일반적으로 계산량이 많으나 매우 사실적인 이미지 생성.
- 전통적으로 영화/광고의 특수 효과 생성 분야에서 널리 사용됨.



Real-Time Rendering

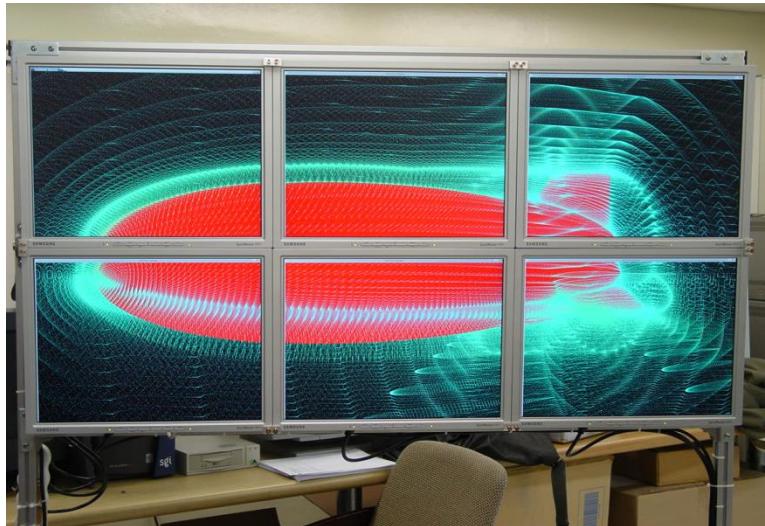
- 대화식(interactive) 그래픽스 응용 소프트웨어를 제작하기 위해서는 '매우 짧은' 시간 안에(예를 들어 1초에 30 프레임) 실시간으로 이미지를 생성해주어야 함.
 - 예: 3차원 게임, 가상 현실, 3D GUI 등
- **과거:** 제한된 시간 안에 렌더링 계산을 마쳐야 하므로 '단순한' 형태의 렌더링 모델 사용 → 생성 이미지의 사실성 저하.
- **현재:** 최근 프로세서 제조 기술의 비약적인 발전으로 인하여 실시간으로 생성할 수 있는 이미지의 사실성이 매우 높아지고 있음.



© 서강대학교 컴퓨터 그래픽스 연구실

Scientific Visualization

- Parallel computing을 통한 방대한 과학적 데이터의 가시화 예



© 서강대학교 컴퓨터 그래픽스 연구실



© U. of Texas at Austin, CCV

Virtual Reality/Augmented Reality/Mixed Reality

- **Virtual reality**
 - Computer technologies that use software to generate the realistic images, sounds and other sensations that replicate a real environment (or create an imaginary setting), and simulate a user's physical presence in this environment.
- **Augmented reality**
 - A live direct or indirect view of a physical, real-world environment whose elements are augmented (or supplemented) by computer-generated sensory input such as sound, video, graphics or GPS data.

From Wikipedia



프로세서 제조 기술의 발전 예: GPU

- 참고 자료
 - ~1993년
 - 2000년 ~ 2001년
 - 2002년 ~ 2003년
 - 2004년 ~ 2005년
 - 2007년
 - 2010년
 - 2011년

~1993년 (SGI 자료)

SiliconGraphics Computer Systems

Client table
July 26, 1993

RealityEngine 192 bits Color 32 bits Z 21" Mon. Std.	VTX 192 bits Color 32 bits Z 19" Mon. Std.	VGXT 48 bits Color 24 bits Z 19" Mon. Std.								
Extreme 24 bits Color 24 bits Z 19" Mon. Std.	Elan 24 bits Color 24 bits Z 19" Mon. Std.	XZ 24 bits Color 24 bits Z 19" Mon. Std.								
XS/XS24 8 bits Color (optional Z) 16" Mon. Std.	XL 8/24 bits Color 15" Mon. Std; PC 16" Mon. Std; SC	Indy 8/24 1.0M X11Lines 400K 3DVect 26K Tmesh	Indy 8/24 1.4M X11Lines 480K 3DVect 40K Tmesh	Indigo Elan 1M 3DVect 270K Tmesh 115K Polygons	Indigo XZ 530K 3DVect 150K Tmesh 58K Polygons	Indigo ² XZ 640K 3DVect 190K Tmesh 70K Polygons	Indigo XS/24 270K 3DVect 76K Tmesh 29K Polygons	Indigo ² XL 1.4M X11Lines 490K 3DVect 40K Tmesh	IRIS Indigo 600K 2DVect 490K 3DVect 40K Tmesh	
Entry Sys. 8 bits Color 16" Mon. Std.	Indy R4000PC/100 MHZ 35 SPECfp92 34 SPECint92	Indy R4000SC/100 MHZ 60 SPECfp92 58 SPECint92	Indigo R4000/100 MHZ 60.5 SPECfp92 58.3 SPECint92	Indigo ² R4000/100 MHZ 60.6 SPECfp92 58.6 SPECint92	Crimson R4000/100 MHZ 61.5 SPECfp92 58.3 SPECint92	Onyx 2xR4400/150 128 MIPS/CPU Desktop	Onyx 4xR4400/150 128 MIPS/CPU Desktop	Onyx 8xR4400/150 128 MIPS/CPU Rack	Onyx 16xR4400/150 128 MIPS/CPU Rack	Onyx 24xR4400/150 128 MIPS/CPU Rack

Performance Graphics

- 3DVects - 10 pixel, connected, 3D, arbitrary orientation.
- Polygons - 10x10 (100 pixel), full 24-bit color, independent, unlighted, Gouraud-shaded, Z-buffered, arbitrary orientation.
- Tmesh - 10x10 (50 pixel) triangle mesh, full 24-bit color, unlighted, flat shaded, Z-buffered, arbitrary orientation.

- 2DVects - 10 pixel, connected, 2D, arbitrary orientation.
- TexAA - 10x10 (50 pixel) Tmesh, anti-aliased, Trilinear Map Mapped, texture mapped.
- TexPix - Texture mapped pixel fill rate using the maximum number of raster boards available & not included in base price
- X11Lines - 10 pixel, connected, 2D vectors

Configuration

	Standard	Options		
	Base Memory	Base Disk	64 MB Memory	Sample Disk
Indy PC	16 MB	--		
Indy SC	16 MB	535 MB		
IRIS Indigo R4000	16 MB	535 MB		
IRIS Indigo ²	16/32 MB	535 MB/1 GB		
IRIS Crimson	16 MB	-		
Onyx	64 MB	-		
POWER CHALLENGE	64 MB	-		
CHALLENGE	64 MB	-		
CHALLENGE M	32 MB	1 GB		

SGI Onyx Graphics Supercomputers



- Processor Data
 - Processors: 2 to 24 MIPS R4400SC 64-bit RISC CPUs
 - Clock frequency: 100MHz / 150MHz
 - Primary caches: 16K / 16K on-chip I/D cache
 - Secondary cache: 1MB combined L2 cache per CPU
- **Graphics Data (RealityEngine2)**
 - **2M t-mesh triangles/second**
 - **930K textured t-mesh triangles/second**
 - **80/160/320M textured, anti-aliased pixels/second**
 - **Hardware texture mapping**
 - **Real-time anti-aliasing**
 - **VGA up to 1600x1200 and HDTV display**
 - **Advanced stereo modes**
 - **Hardware image processing acceleration**
 - **48-bit RGBA color, quad-buffered (192 bits total)**
 - **256 to 1024 bits per pixel**
 - **40 to 160MB frame buffer**
 - **NTSC/PAL/S-Video output**



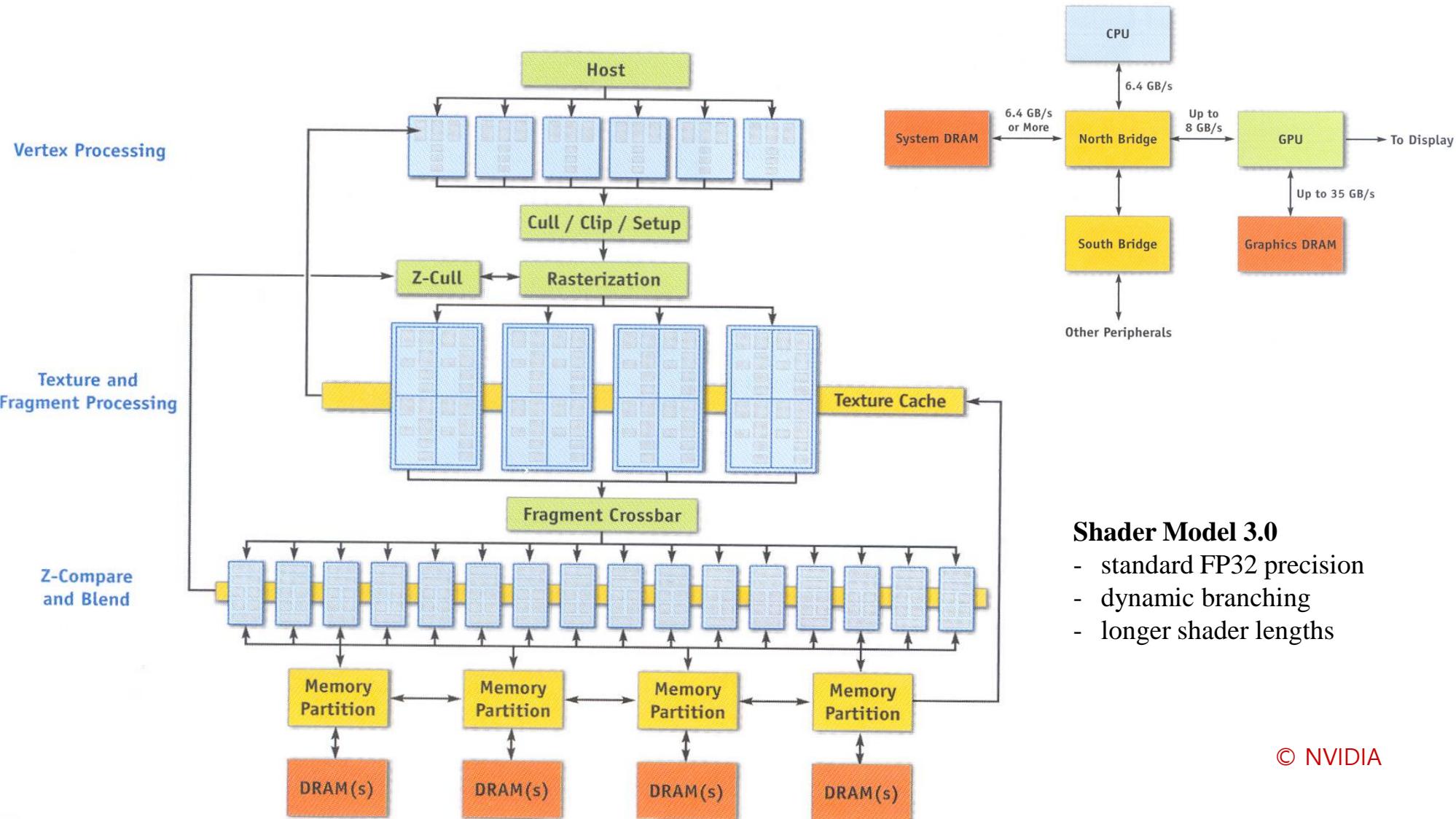
2000년 ~ 2001년

	Xbox	Sony PlayStation 2	Nintendo Game Cube
Graphics Processor	250 MHz Custom-designed chip by MS and NVIDIA	147.456 MHz	202.5 MHz Custom chip “Flipper”
Total memory	64 MB	32 MB	43 MB
Memory Bandwidth	6.4 GB/s	3.2 GB/s	3.2 GB/s
Polygon Performance	125 M/s	66 M/s	6-12M /s
Simultaneous Textures	4	1	N/A
Pixel Fill Rate – No Texture	4.0 G/s	2.4 G/s	N/A
Pixel Fill Rate – 1 Texture	4.0 G/s	1.2 G/s	N/A
Pixel Fill Rate – 2 Texture	4.0 G/s	0.6 G/s	N/A

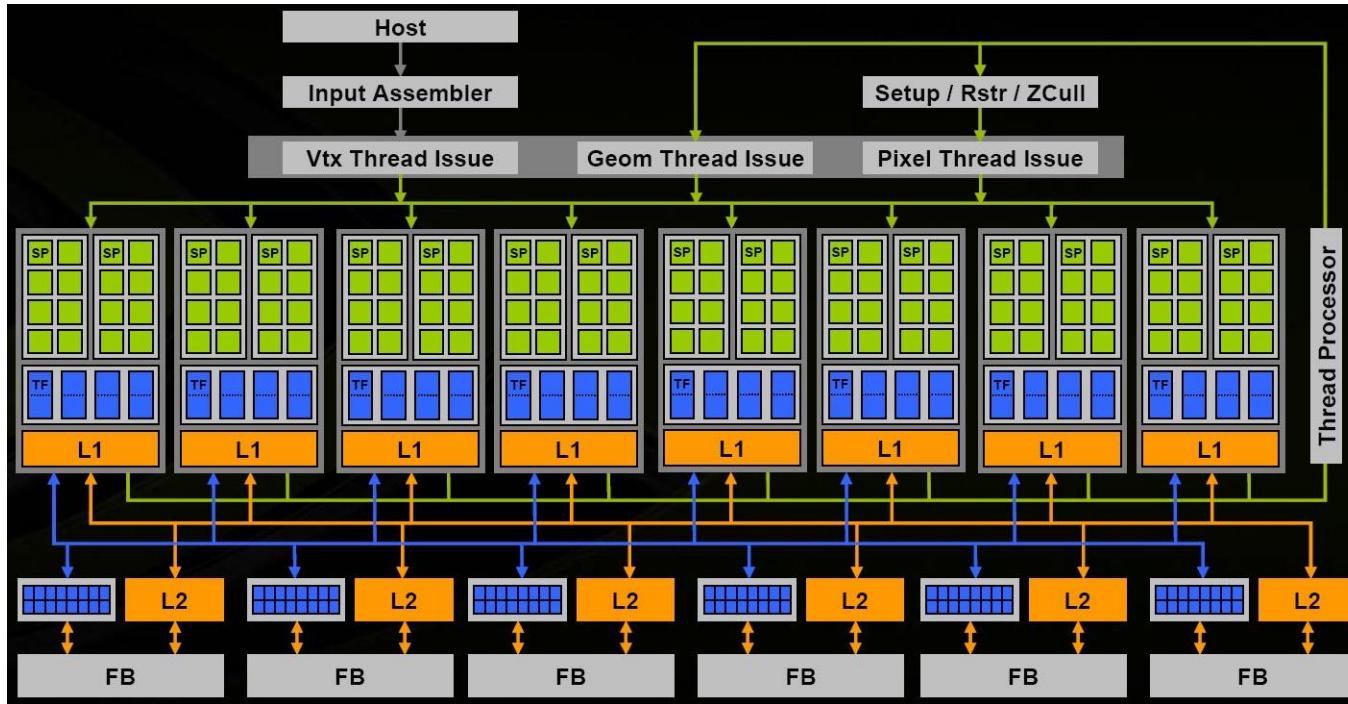
2002년 ~ 2003년

	ATI Radeon 9700 PRO	NVIDIA GeForce4 Ti4800	NVIDIA GeForceFX 5800	NVIDIA GeForceFX 5800 Ultra
Chip Technology	256-bit	256-bit	256-bit	256-bit
Transistors	~107 Million	63 Million	125 Million	125 Million
Memory Bus	256-bit DDR	128-bit DDR	128-bit DDR2	128-bit DDR2
Memory Bandwidth	19.8GB/s	10.4GB/s	12.8GB/s	16GB/s
Pixel Fill Rate	2.6 Gigapixels/s	1.24 Gigapixels/s	~3.2Gigapixels/s	~4 Gigapixels/s
Anti-Aliased Fill Rate	15.6 Billion AA Samples/s	4.8 Billion AA Samples/s	~12.8 Billion AA Samples/s	~16 Billion AA Samples/s
Triangle Transform Rate	325M Triangles/s	69M Triangles/s	280M Triangles/s	350M Triangles/s
Graphics Memory	128/256MB	128MB	128/256MB	128/256MB
GPU Clock	325 MHz	300 MHz	~400 MHz	~500 MHz
Memory Clock	310 MHz (620 DDR)	325 MHz (650 DDR)	400 MHz (800 DDR2)	500 MHz (1000 DDR2)
Textures per Texture Unit	8	4	16	16

2004년: GeForce 6 Series (NV40) - 6800 Ultra



2007년: GeForce 8 Series (G80) - 8800 GTX



© NVIDIA

Unified shader architecture

- 16 highly threaded streaming multiprocessors: 128 Cores
- Thousands of threads per applications
- 367 GFLOPS
- 768MB DRAM
- 86.4GB/s memory bandwidth
- 4GB/s bandwidth to CPU

Rendering Pipeline의 부하 분석 예

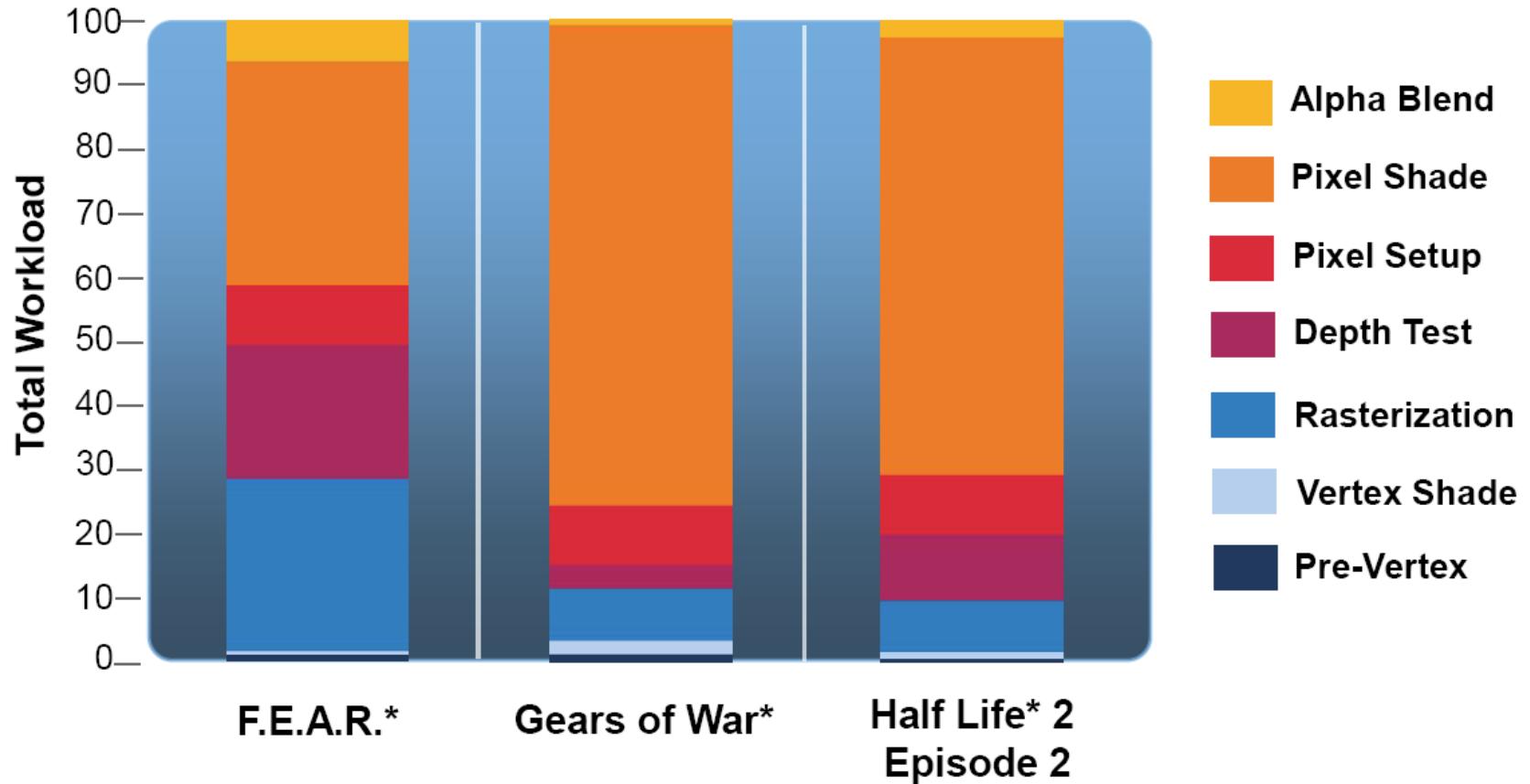
Why unify?



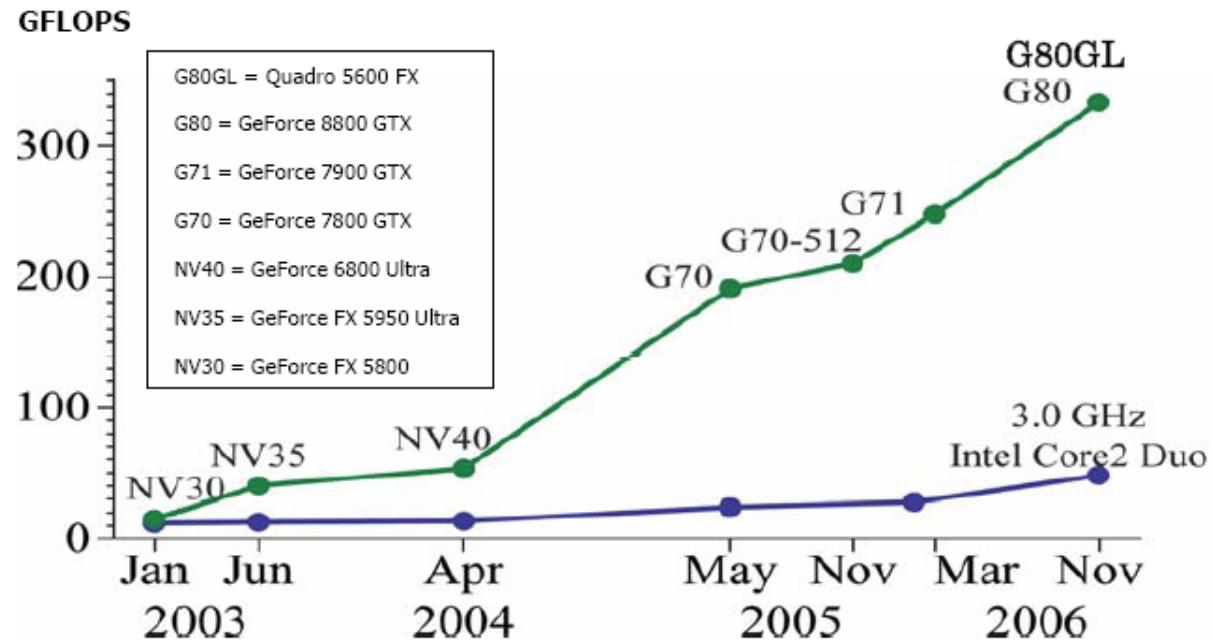
Figure 15. Fixed shader performance characteristics

From Technical Brief: NVIDIA GeForce 8800 GPU Architecture Overview, 2006.

과연 각 단계가 application, geometry, 그리고 rasterizer stage 중 어느 단계에 해당할까?



From Seiler et al., "Larrabee: A many-core x86 architecture for visual computing," SIGGRAPH 2008.



3.0GHz Intel Core2 Duo CPU
 f-op: 32GFLOPS
 m. bw: 8.4GB/s

NVIDIA GeForce 8800GTX GPU
 f-op: 367GFLOPS
 m. bw: 86.4GB/s

Figure 1-1. Floating-Point Operations per Second for the CPU and GPU

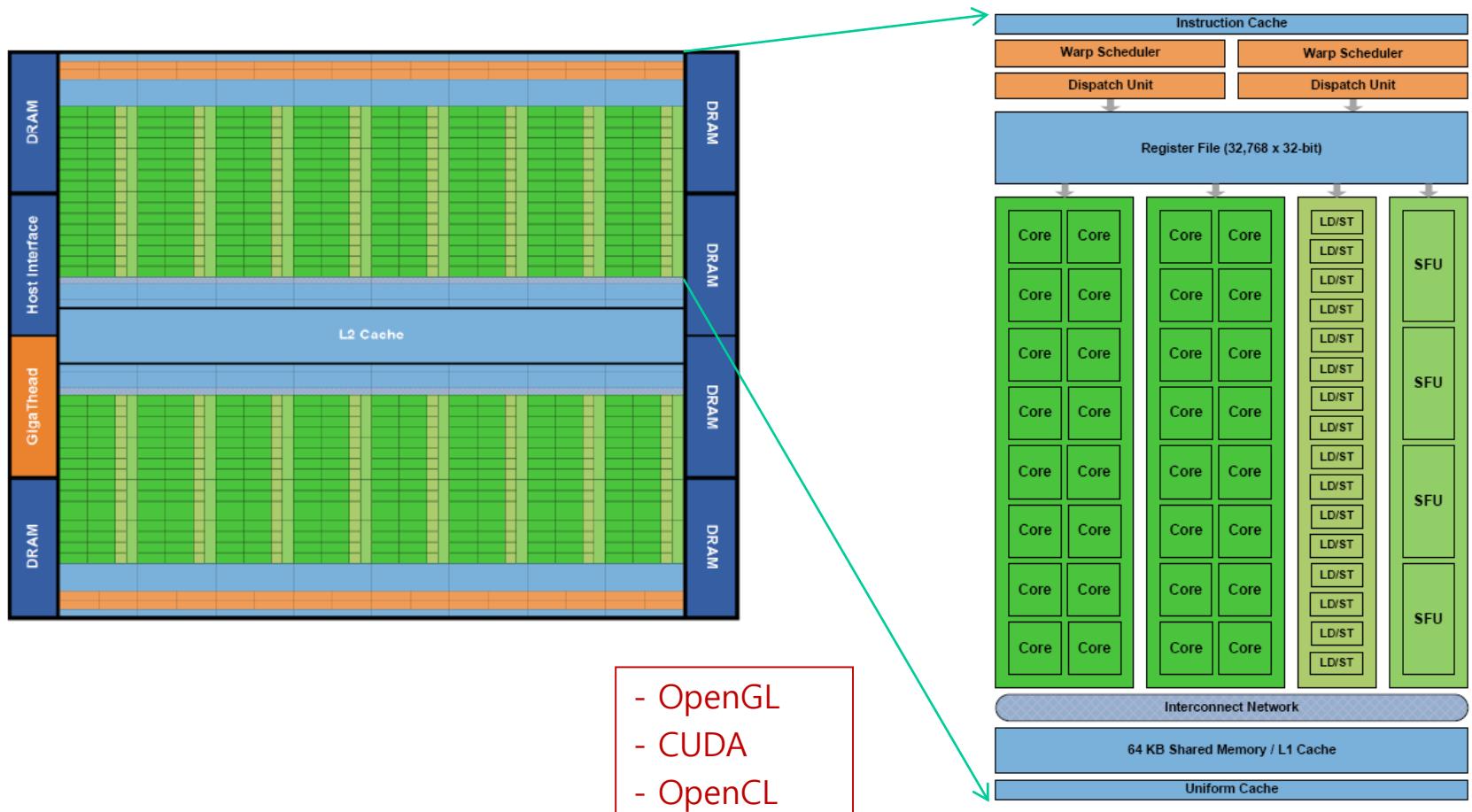
© NVIDIA

2010년: GeForce 500 Series (GF100) - GTX 580

- Fermi architecture

- 16 Streaming Multiprocessors with 32 Streaming Processors each → **512 Streaming Processors**
- **37.06 Gpixels/s**
- **1581.1 GFLOPS**

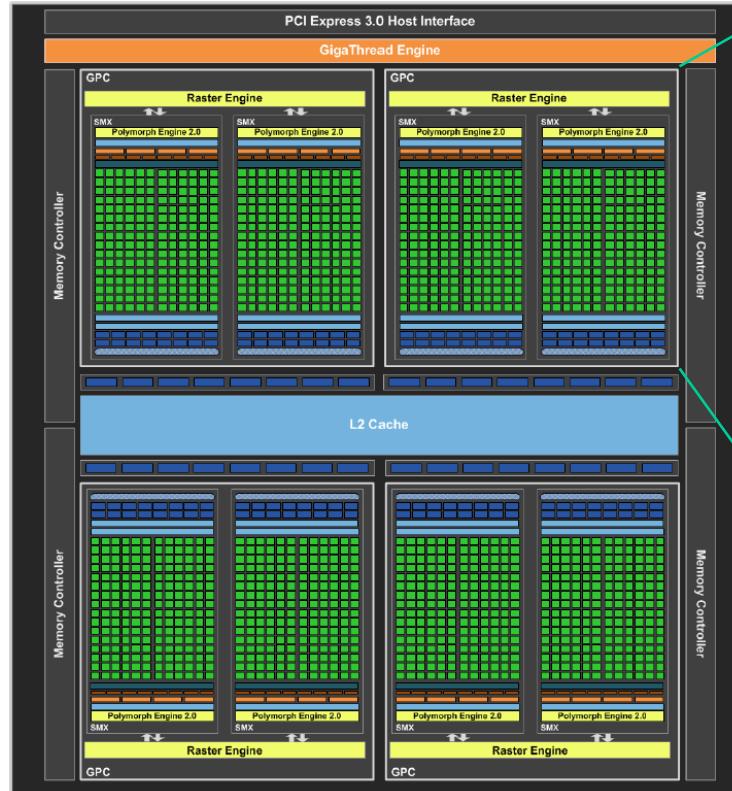
© NVIDIA



2012년: GeForce 600 Series (GK104) - GTX 680

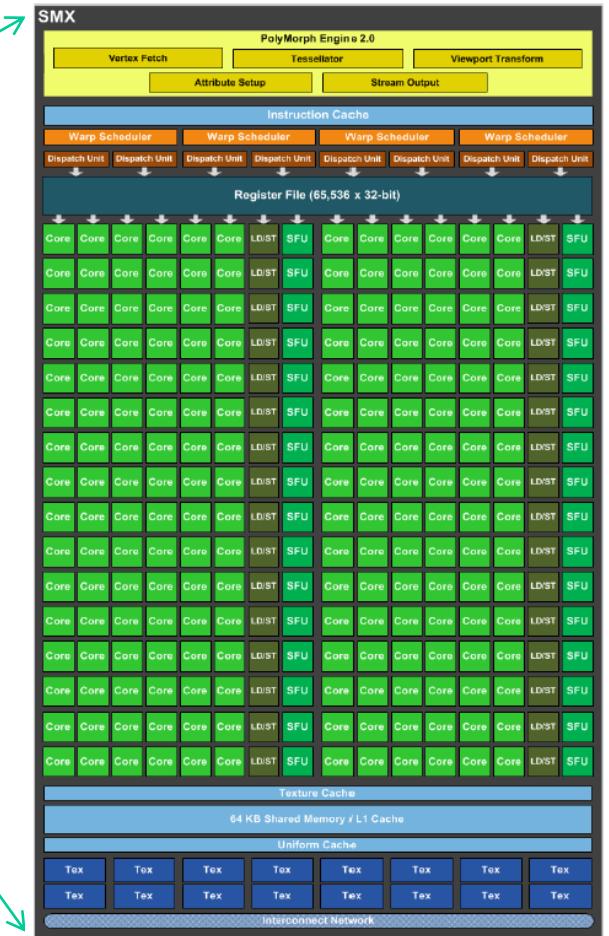
- Kepler Architecture (GK104)
 - 4 Graphics Processing Clusters (GPC)
 - 8 next-generation Streaming Multiprocessors (SMX): 1,536 CUDA Cores
 - Four memory controllers

© NVIDIA



128.8 Gigatexels/sec

3090 GFLOPS



2017년: NVIDIA GeForce GTX 1080 (GP104)



2560 CUDA Cores
8873 GFLOPS
160 Texture units
64 ROPs
277.3 Gigatexels/sec

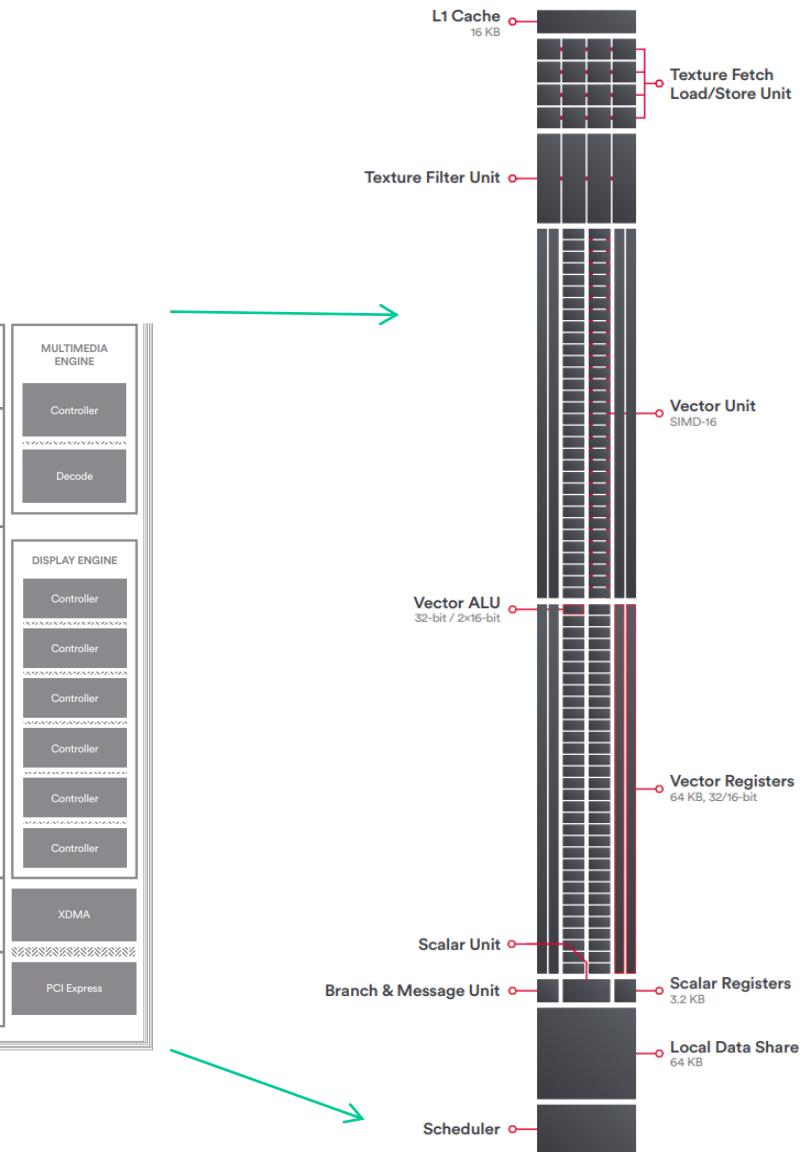
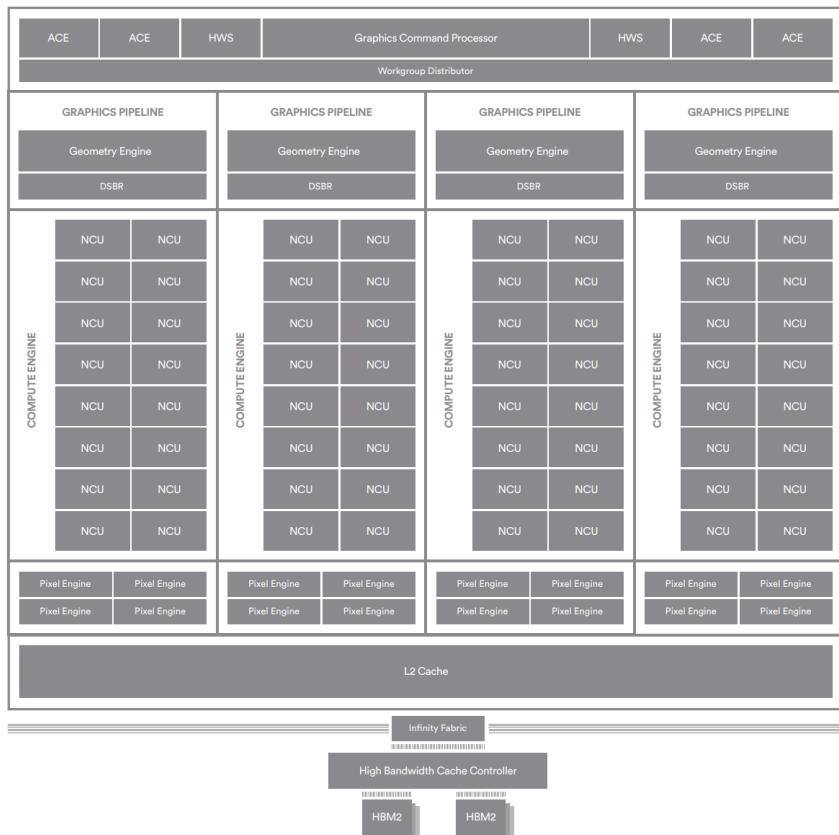
2018년: GeForce RTX 2080 Ti

Model	Launch	Code name	Process	Transistors (billion)	Die size (mm ²)	Core config ^[a]	Bus interface	L2 Cache (MiB)	Clock speeds		Memory			Fillrate ^[b]		Processing power (GFLOPS) ^[b]				Ray-tracing Performance		TDP (Watts)	NVLink support	Release price (USD)			
									Base core clock (MHz)	Boost core clock (MHz)	Memory (MT/s)	Size (GiB)	Bandwidth (GB/s)	Bus type	Pixel (GP/s) ^[c]	Texture (GT/s) ^[d]	Single precision	Double precision	Half precision	Tensor compute (FP16)	Rays/s (Billions)	RTX OPS/s (Trillions)	MSRP	Founders Edition			
GeForce RTX 2060 ^[88]	January 15, 2019	TU106-200-KA-A1	TSMC 12FFN	10.8	445	1920:120:48:240:30 (30) (3)	PCIe 3.0 x16	3	1365	1680	14000	6	336.0	GDDR6	192	65.52 80.64	163.80 201.60	5 241.60 6 451.20	163.80 201.60	10 483.20 12 902.40	41 932.80 51 609.60	5	37	160	No	\$349	
GeForce RTX 2070 ^{[89][90]}	October 17, 2018	TU106-400-A1				2304:144:64:288:36 (36) (3)			1410	1620		8	448.0		90.24 103.68	203.04 233.28	6 497.28 7 464.96	203.04 233.28	12 994.56 14 929.92	51 978.24 59 719.68	6	42	175	\$499	\$599		
GeForce RTX 2080 ^{[91][92]}	September 20, 2018	TU104-400-A1		13.6	545	2944:184:64:368:46 (46) (6)			1515	1710					256	96.96 109.44	278.76 314.64	8 920.32 10 068.48	278.76 314.64	17 840.64 20 136.96	71 362.56 80 547.84	8	57	215	\$699	\$799	
GeForce RTX 2080 Ti ^[93]	September 27, 2018	TU102-300-K1-A1		18.6	754	4352:272:88:544:68 (68) (6)			1545			11	616.0		352	118.80 135.96	367.20 420.24	11 750.40 13 447.68	367.20 420.24	23 500.80 26 895.36	94 003.20 107 581.44	10	76	250	2-way NVLink	\$999	\$1199
Nvidia TITAN RTX ^[94]	December 18, 2018	TU102-400-A1				4608:288:96:576:72 (72) (6)			1350	1770 ^[e]		24	672.0		384	129.60 169.92	388.80 509.76	12 441.60 16 312.32	388.80 509.76	24 883.20 32 624.64	99 532.80 130 498.56	11	84	280		N/A	\$2,499

~13.4 TFLOPS!!!

AMD Radeon RX Vega64 Liquid Cooled Edition

- 64 next-generation compute units (NCUs)
- Each NCU has 64 stream processors.
- A total of 4,096 stream processors
- 13.7 TFLOPS



© AMD

Supercomputer와 성능 비교: FLOPS (FLoating-point OPerations per Second)

1993	Thinking Machines CM-5/1024	65.5 GFLOPS	DoE-Los Alamos National Laboratory; National Security Agency
	Fujitsu Numerical Wind Tunnel	124.50 GFLOPS	National Aerospace Laboratory, Tokyo, Japan
	Intel Paragon XP/S 140	143.40 GFLOPS	DoE-Sandia National Laboratories, New Mexico, USA
1994	Fujitsu Numerical Wind Tunnel	170.40 GFLOPS	National Aerospace Laboratory, Tokyo, Japan
1996	Hitachi SR2201/1024	220.4 GFLOPS	University of Tokyo, Japan
	Hitachi/Tsukuba CP-PACS/2048	368.2 GFLOPS	Center for Computational Physics, University of Tsukuba, Tsukuba, Japan
1997	Intel ASCI Red/9152	1.338 TFLOPS	DoE-Sandia National Laboratories, New Mexico, USA
1999	Intel ASCI Red/9632	2.3796 TFLOPS	
2000	IBM ASCI White	7.226 TFLOPS	DoE-Lawrence Livermore National Laboratory, California, USA
2002	NEC Earth Simulator	35.86 TFLOPS	Earth Simulator Center, Yokohama, Japan

© Wikipedia

3D Computer Graphics Needs High Performance!

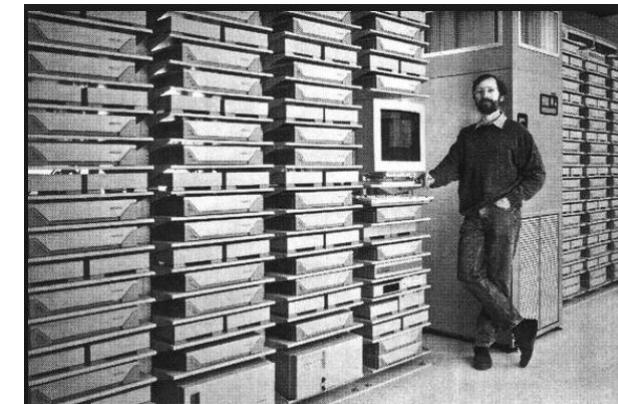
- 영화 Toy Story 관련 자료 (1990년대 초반) -

- Characters: 76
- Objects: 366
- Lines of code describing the objects: 4.5million
- Hairs on Andy's head: 12,384
- Hairs on Sid's head: 15,977
- Animation controls on Sid's backpack: 128
- Storyboard drawings: 25,000
- Number of people at Pixar working on the film: 110
- Number of animator: 28
- Number of technical director: 30
- Leaves on a typical tree in Andy's neighborhood: 10,000
- Trees on Andy's block: 100+
- Leaves on Andy's block: 1.2 million
- Number of minutes: about 75

- Resolution per frame: 1526x922
- Total storage for final frames: over 500GB
- # of basic arithmetic operations per pixels: 500,000
- Amount of RenderMan data files to be sent thru the renderer: 34TB
- Painted texture maps: 2000+
- RenderMan Shaders: 1300
- Used 300 Sun Sparc20's



© Pixar



★ 과연 이러한 정도 이상의 렌더링 작업을 실시간으로 수행하면서 영화를 상영할 수 있을까?

GPGPU (General-Purpose Computing on GPU)

- “**General-purpose computing on graphics processing units (GPGPU)** is the means of using a graphics processing unit (GPU), which typically handles computation only for computer graphics, to perform computation in applications traditionally handled by the central processing unit (CPU).” (*from Wikipedia*)
- 최근 그래픽스 프로세서는 massively-parallel streaming processing을 위한 프로세서로서 비약할 만한 성능 향상을 이루었으며, 그러한 추세는 계속될 것으로 예상됨.
 - “A *stream* is simply a set of records that require similar computation. Streams provide SIMD parallelism. *Kernels* are the functions that are applied to each element in the stream.” (*from Wikipedia*)
- 최근 GPU의 놀라운 성능 향상으로 인하여 그래픽스 분야의 문제 뿐만 아니라 “compute-intensive data-parallel” 성질을 가지는 **일반 응용 문제를 해결**하는데 유용하게 쓰이고 있음.

Programming with CUDA, OpenCL, and RenderScript

3D Computer Graphics on Mobile Devices

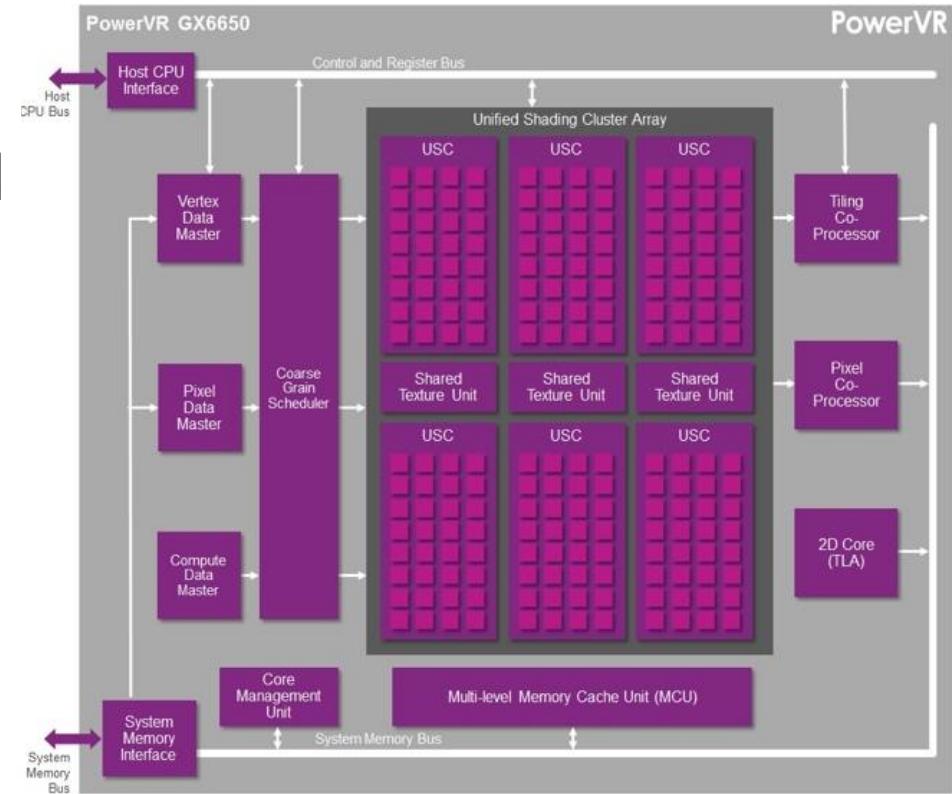
2014: Imagination PowerVR GX6650

Monday, February 24, 2014, 05:51 am PT (08:51 am ET)

Imagination unveils 192-core mobile GPU potentially bound for future Apple iPhones, iPads

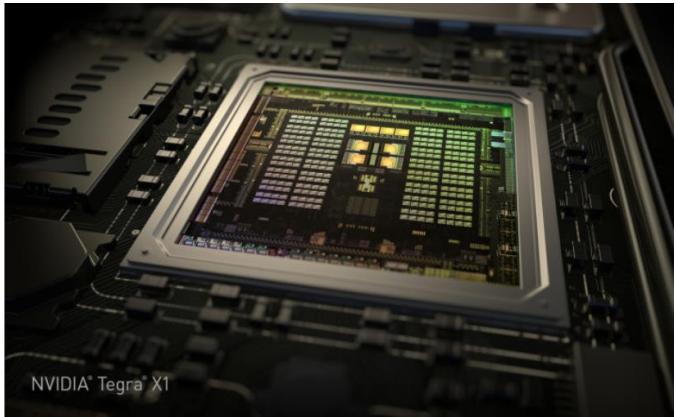
By [AppleInsider Staff](#)

Imagination Technologies, the company whose PowerVR graphics processors are a key component of Apple's iOS devices, unveiled its latest creation on Monday: a 192-core GPU that it claims will produce the most powerful graphics yet in mobile phones and tablets.



2015년: NVIDIA Tegra X1

© NVIDIA



- **CPU:** ARMv8 ARM Cortex-A57 quad-core + ARM Cortex-A53 quad-core (64-bit)
- **GPU:** Maxwell-based **256 core GPU**
- MPEG-4 HEVC & VP9 encoding/decoding support
- TSMC 20 nm process
- Power consumption less than 10 Watts

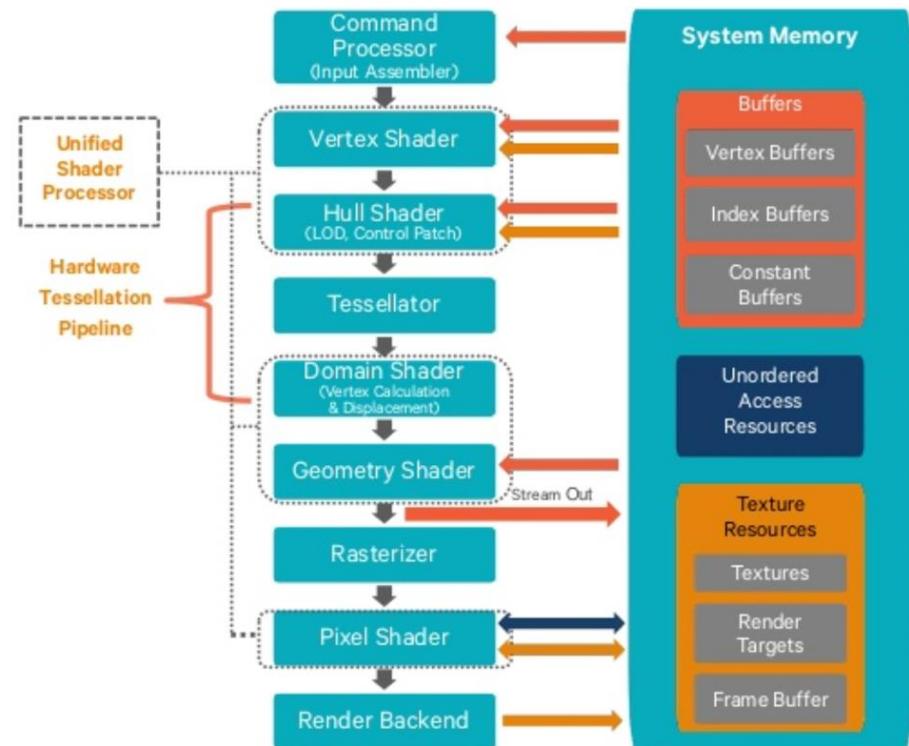
GPU	Tegra K1 (Kepler GPU)	Tegra X1 (Maxwell GPU)
SMs	1	2
CUDA Cores	192	256
GFLOPs (FP32) Peak	365	512
GFLOPs (FP16) Peak	365	1024
Texture Units	8	16
Texel fill-rate	7.6 Gigatexels/sec	16 Gigatexels/sec
Memory Clock	930 MHz	1.6GHz MHz
Memory Bandwidth	14.9 GB/s	25.6 GB/s
ROPs	4	16
L2 Cache Size	128KB	256KB
Manufacturing Process	28-nm	20-nm
Z-cull	256 pixels/clock	256 pixels/clock
Raster	4 pixels/clock	16 pixels/clock
Texture	8 bilinear filters/clock	16 bilinear filters/clock
ZROP	64 samples/clock	128 samples/clock

You Will Learn What These Mean:

Name	Microarchitecture		Clock (MHz)	Fillrate			GFLOPS	API (version)					Used in Qualcomm...	References
	Type	ALUs [note 1]		Fab (nm)	MTriangles/s	Pixel (GP/s)		OpenGL ES	OpenVG	OpenCL	OpenGL	Direct3D		
1xx series														
Adreno 420	Qualcomm model 5-way VLIW ^[1]	128	28	500/600	281.3/337.5	4/4.8	144/172.8	3.1	1.1	1.2 full profile	N/A	(feature level 11_1) ^[5]	Snapdragon 805 (APQ8084)	[6]
Adreno 430		?	20	500/600	?	?	324/388.8						Snapdragon 810 (APQ8094, MSM8994)	

Adreno 420 architectural improvements

- DX11.2 3D pipeline
 - Hardware tessellation
 - Geometry shading
 - Stream out from VS, DS, GS
 - Programmable blending
- Upgraded compute
 - Direct compute, OpenCL 1.2 Full profile
 - Faster RenderScript
- Improved texturing
 - Improved texture performance
 - Support for higher level texture filtering (e.g., Aniso) with less performance impact
 - ASTC support, better LOD & filtering quality
 - Larger caches: texture cache, L2 cache
- Improved ROPs & Z
 - Faster depth rejection
 - Designed to achieve peak draw rate more often



Adreno 420 supports most advanced graphics APIs

Feature/APIs	OpenGL ES 3.0	OpenGL ES 3.1	Android Extension Pack
Compute Shader	No	Yes	Yes
Atomics	No	Yes	Yes
Image Load/Store	No	Yes	Yes
Draw Indirect	No	Yes	Yes
Texture Gather	No	Yes	Yes
Multisample Textures	No	Yes	Yes
Stencil Textures	No	Yes	Yes
Separate Shader Objects	No	Yes	Yes
Advanced Blending Modes (Programmable Blending)	No	Yes	Yes
Geometry Shaders	No	No	Yes
Tessellation Shaders	No	No	Yes

[CSE4170 기초 컴퓨터 그래픽스]

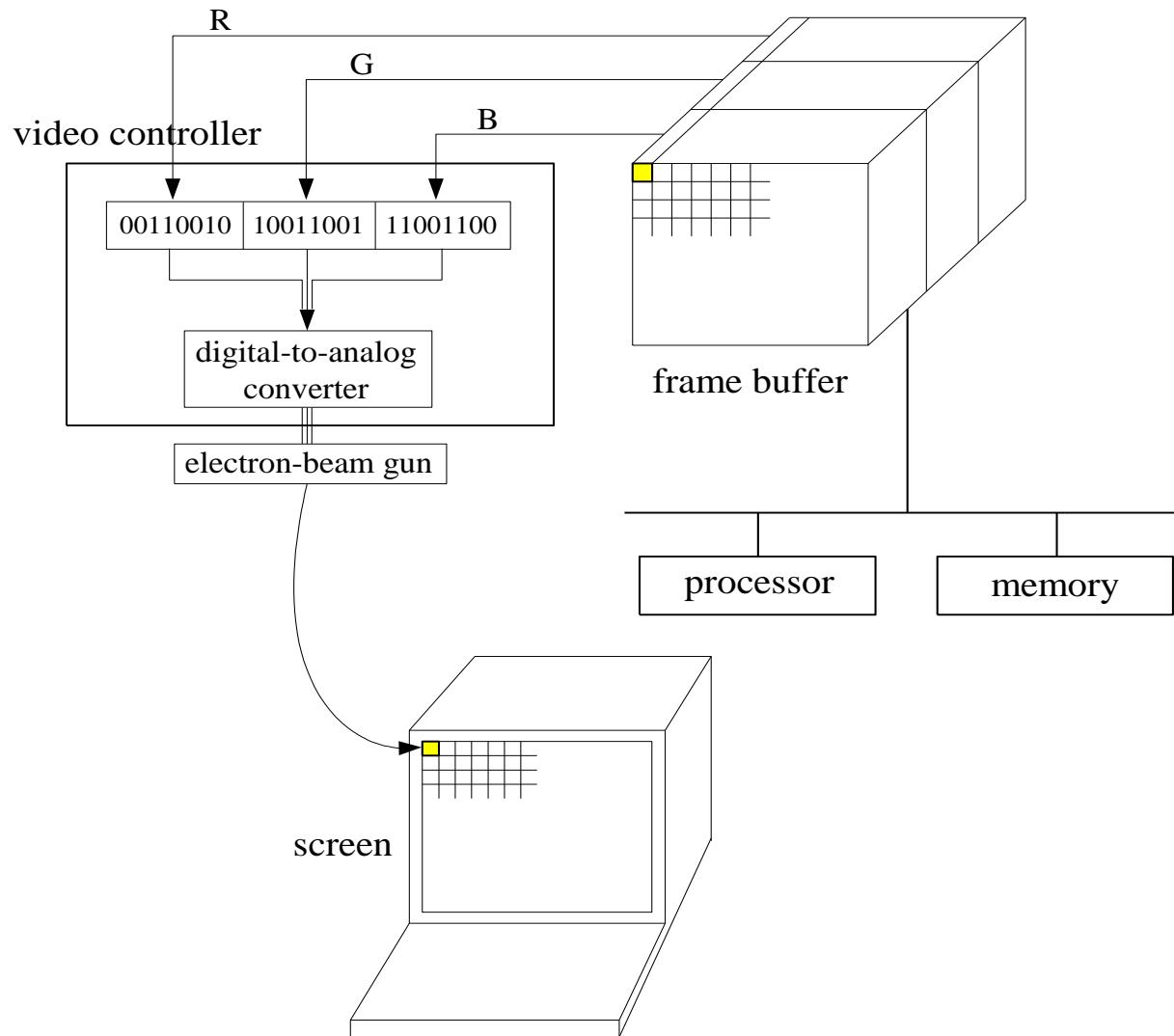
2019년도 1학기

강의자료 I

Introduction to Raster Graphics System

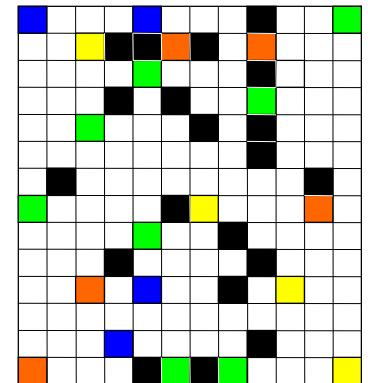
Introduction to 3D Raster Graphics System

Raster Graphics System



Raster Image

- 직사각형 형태의 이미지 영역을 화소(pixel)라 부르는 조그마한 영역으로 나누어, 각 화소를 해당 영역을 대표하는 색깔로 칠함.
- '연속적인' 영상을 유한개의 화소를 사용하는 영역으로 표현하므로 오차가 발생함. → aliasing
- 화소(pixel = picture element)
- 해상도(resolution)
 - 640X480, 1024X768, 1280X1024, ...
 - 해상도가 높으면 보다 정밀한 이미지를 표현할 수 있으나 필요로 하는 메모리의 양이 증가
- 벡터 이미지(vector image)와의 비교





64X64

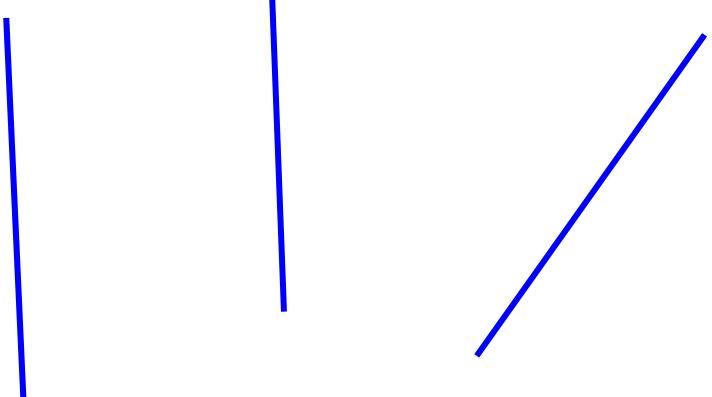
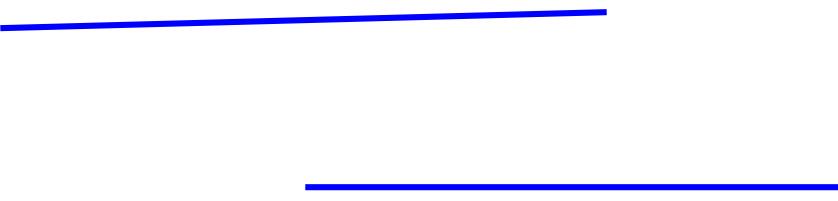


128X128



256X256

Jagged edge 현상



Color Model

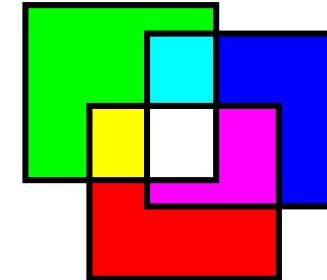
- 색깔(color)을 어떠한 모델을 사용하여 수치적으로 표현할 것인가?
 - RGB color model
 - CMY color model
 - HSV color model
 - 기타

RGB Color Model

- 주어진 색깔을 빨간색(Red), 초록색(Green), 파란색(Blue) 등의 세 가지 원색을 (검정 바탕에) 적절히 더해 표현하는 방식 → Additive color model

- Color tuple (r , g , b) ($0.0 \leq r, g, b \leq 1.0$)로 표현

```
glClearColor(1.0, 1.0, 1.0, 1.0);  
	glColor3f(0.765, 0.471, 0.686);
```



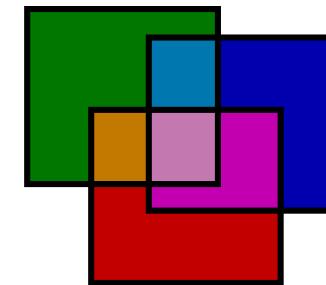
R = 1.0
G = 1.0
B = 1.0

- Primary colors

빨간색(Red) = (1, 0, 0), 초록색(Green) = (0, 1, 0), 파란색(Blue) = (0, 0, 1) : additive primary colors
시안색(Cyan) = (0, 1, 1), 자홍색(Magenta) = (1, 0, 1), 노란색(Yellow) = (1, 1, 0) : subtractive primary colors
검정색(black) = (0, 0, 0), 흰색(White) = (1, 1, 1), 회색(Gray) = (a, a, a) : neutral colors

- 기타 색깔

- (127, 255, 212) = aquamarine
 - (0.196, 0.6, 0.8) = ???
 - (0.435, 0.259, 0.259) = ???



R = 0.765
G = 0.471
B = 0.686

- ★ RGB 색깔 모델은 직관적이지 않으나 기계적으로

편리하게 다룰 수 있어 디스플레이 시스템 등에서 널리 쓰일 뿐만 아니라 컴퓨터 그래픽스 분야에서 근간을 이루는 색깔 모델임.

Computer Representation for RGB Color Model

- r, g, b 각 채널 값을 유한 개의 비트를 사용하여 표현
 - 만약 채널당 8비트, 즉 한 색깔 당 총 24비트를 사용할 경우
 - 0과 1사이의 채널 값을 256 단계로 샘플링하여 표현 (0/255, 1/255, 2/255, ..., 254/255, 255/255)
 - 이 경우 색깔을 (195, 120, 175)와 같이 나타냄.
 - 각 채널 당 2^8 가지의 서로 다른 값을 나타낼 수 있으므로 총 2^{24} 개의 서로 다른 색깔을 표현 할 수 있음. → 약 1670만개의 색깔 표현 가능
 - 만약 한 색깔당 총 16 비트를 사용할 경우
 - 2^{16} , 즉 약 6만5천개의 색깔 표현 가능
 - 만약 한 색깔당 총 n 비트를 사용할 경우
 - ???

Computer Representation for RGB Color Model



R

G

B



- r, g, b 각 채널 값을 유한 개의 비트를 사용하여 표현
 - 만약 채널당 8비트, 즉 한 색깔 당 총 24비트를 사용할 경우
 - 0과 1사이의 채널 값을 256 단계로 샘플링하여 표현 (0/255, 1/255, 2/255, ..., 254/255, 255/255)
 - 이 경우 색깔을 (195, 120, 175)와 같이 나타냄.
 - 각 채널 당 2^8 가지의 서로 다른 값을 나타낼 수 있으므로 총 2^{24} 개의 서로 다른 색깔을 표현 할 수 있음. → 약 1670만개의 색깔 표현 가능
 - 만약 한 색깔당 총 16 비트를 사용할 경우
 - 2^{16} , 즉 약 6만5천개의 색깔 표현 가능
 - 만약 한 색깔당 총 n 비트를 사용할 경우
 - ???

- ★ 채널 당 더 많은 비트를 사용할 수록 메모리 요구량이 증가함.
 - 만약 해상도가 1280X1024인 래스터 이미지를 24비트 색깔을 사용할 경우
 $1280 \times 1024 \times 24 / 8B = 3.75MB$ 의 메모리 필요함.

- ★ 일반적으로 채널 당 8비트면 충분하나 정밀한 그래픽스 계산 또는 이미지 처리를 위해서 채널당 12/16/24/32 비트 등을 사용하기도 함.

Two Different Resolutions of Raster Images

- ❶ 주어진 이미지를 얼마나 많은 화소를 사용하여 표현할 것인가?
 - 640X480, 1024X768, 1280X1024, ...
- ❷ 얼마나 많은 색깔로 구성된 팔레트(palette)에서 원하는 색깔을 뽑아 각 화소를 칠할 것인가?
 - 24비트 색깔, 16비트 색깔, 8비트 색깔, ...
- 이 두 가지 해상도에 따라 하나의 래스터 이미지를 저장하는데 필요한 메모리의 양이 결정됨.
- **비트맵(bitmap)** 또는 **픽스맵(pixmap)**
 - 래스터 이미지 데이터
 - 종종 비트맵은 흑백 이미지의 픽스맵에 한정하여 쓰이기도 함.

RGB Mode and Color Index Mode

- **RGB mode**

- Full color mode, direct color mode, ...
- 각 화소 값이 그 화소에 칠할 (r, g, b) 값을 의미

n_R : # of bits for R channel

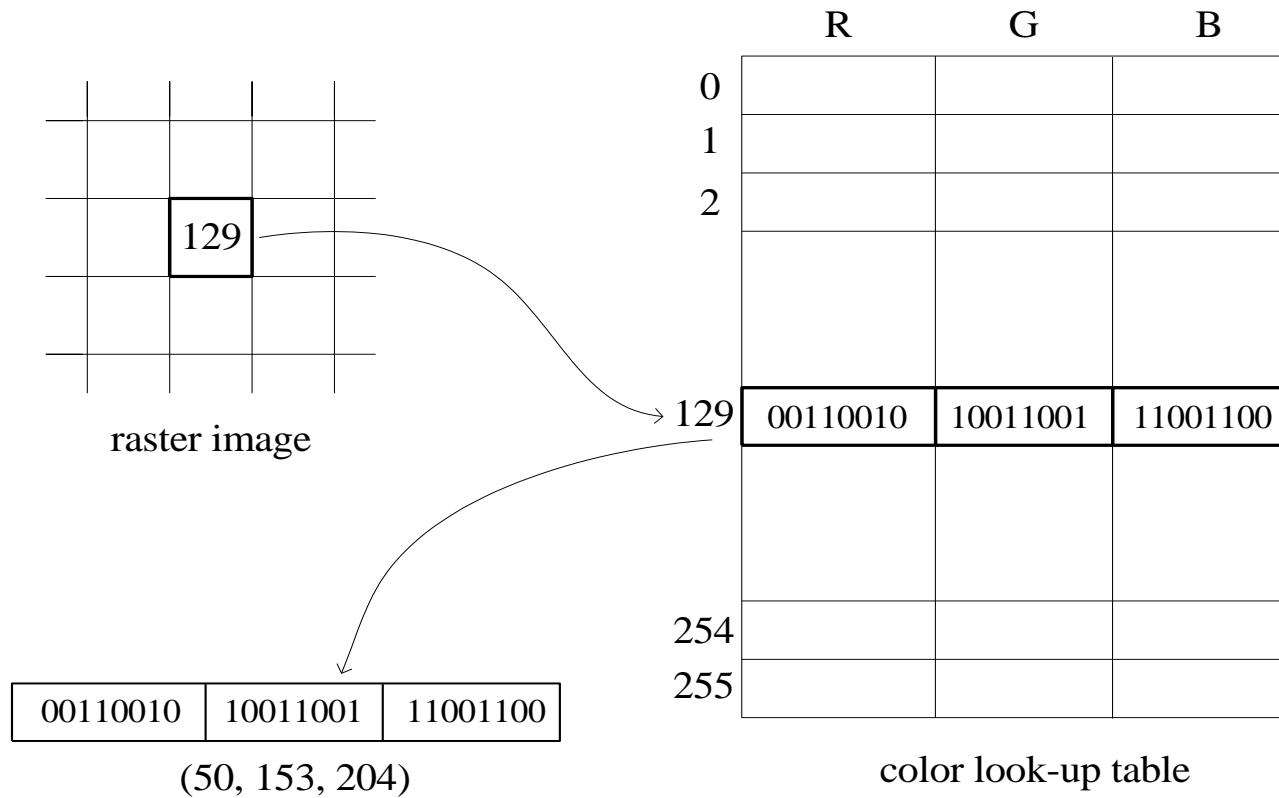
n_G : # of bits for G channel

n_B : # of bits for B channel

$$n = n_R + n_G + n_B$$

$$2^n = 2^{(n_R+n_G+n_B)} \text{개의 색깔 표현 가능}$$

- **Color-index mode**



m_R : # of bits for R channel of color map

m_G : # of bits for G channel of color map

m_B : # of bits for B channel of color map

$m = m_R + m_G + m_B$: color map width

n : raster image depth



n : How many colors can be used to paint a raster image?

m : From how many colors can we select color for each pixel?

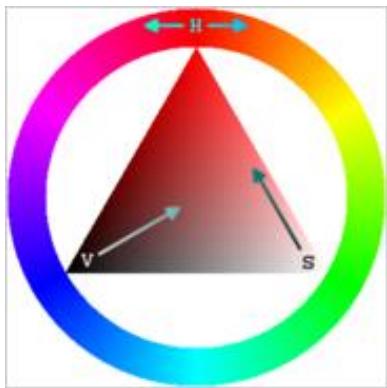
CMY (CMYK) Color Model

- 주어진 색깔을 빨간색(Red), 초록색(Green), 파란색(Blue) 등의 세 가지 원색을 (흰색에서) 적절히 빼서 표현하는 방식 → Subtractive color model
 - (c, m, y) ($0.0 \leq c, m, y \leq 1.0$)로 표현
 - $(c, m, y) = (1, 1, 1) - (r, g, b)$
- ★ CMY 색깔 모델은 주로 출력 장치에 쓰임.
★ 출력장치에서 무채색을 효과적으로 생성해내기 위해 보통 검정색(black)을 더해 CMYK model이 사용함.

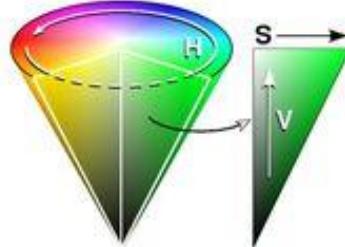
HSV (or HSB) Color Model

- 주어진 색깔을 Hue(H), Saturation(S), Value(V) 등의 세 가지 직관적인 값으로 표현하는 방식

© Wikipedia



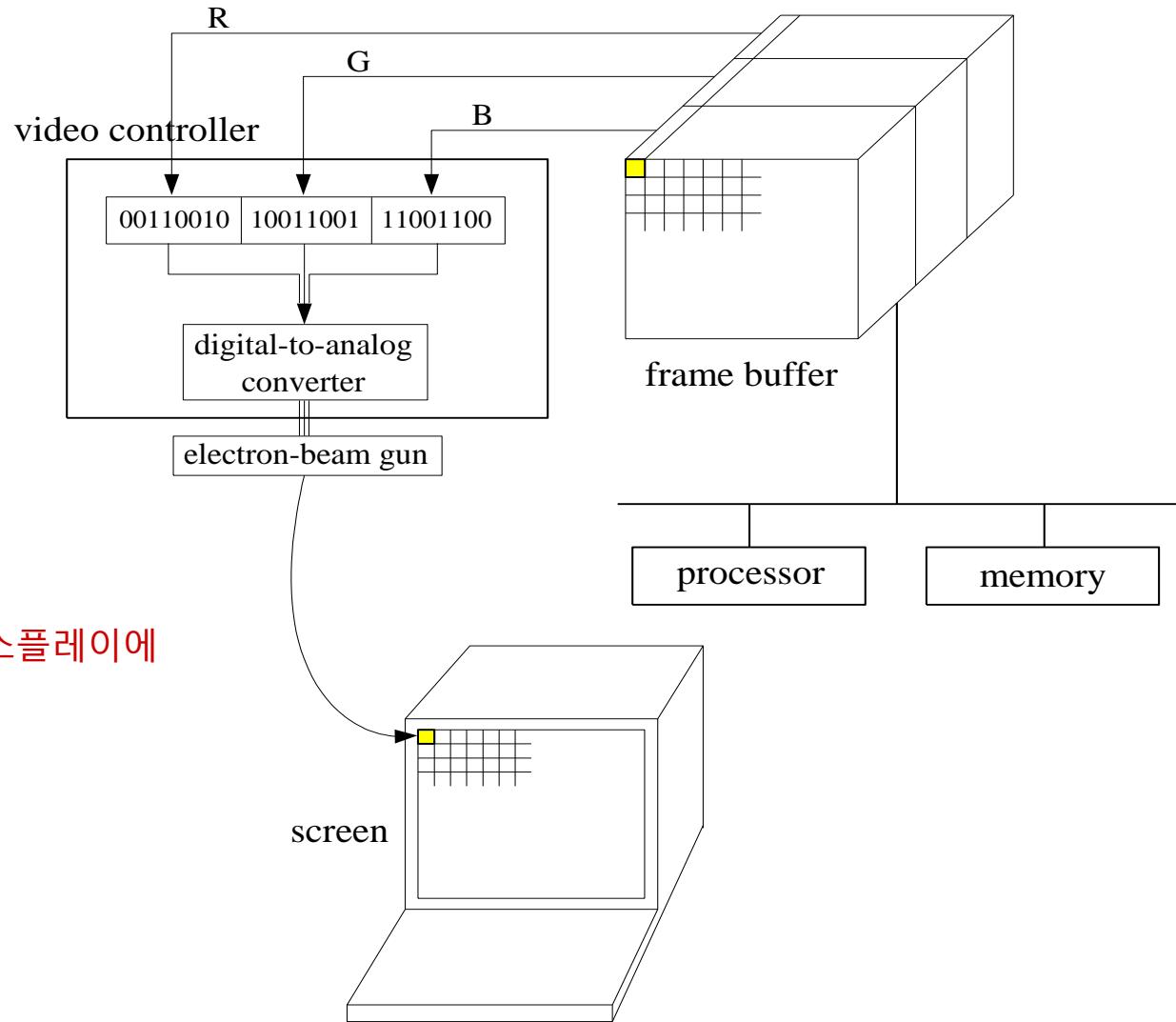
HSV color wheel



a conical representation

- (r, g, b) , (c, m, y) , (h, s, v) 값 간에 어떤 방식으로 변환이 가능할까?

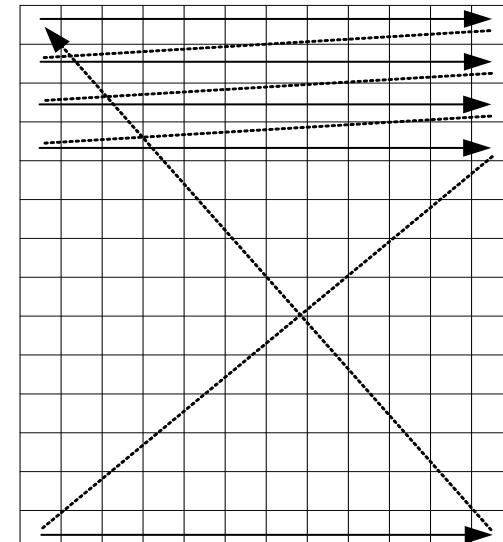
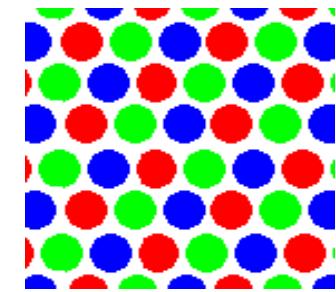
Raster Graphics System



래스터 이미지 또는 래스터 디스플레이에
기반을 둔 그래픽스 시스템

관련 기본 용어

- 화면(screen)
 - 주어진 해상도를 가지는 래스터 이미지 도시
- 프레임 버퍼(frame buffer)
 - 항상 한순간에 화면에 도시되는 래스터 이미지에 대한 픽스맵 데이터는 프레임 버퍼라는 특수한 메모리에 저장이 되어 있어야 함.
 - 비디오 메모리(video memory)
- 프레임 버퍼의 깊이(depth)
- 해상도와 프레임 버퍼 크기와의 관계
- 비디오 제어기(video controller)
- 디지털 아날로그 변환기(digital-to-analog converter)
- CRT 모니터의 형광체(phospher)
- 래스터 스캔(raster scan)
- 스캔 라인(scan line)
- 리프레쉬 속도(refresh rate)



래스터 스캔의 순서

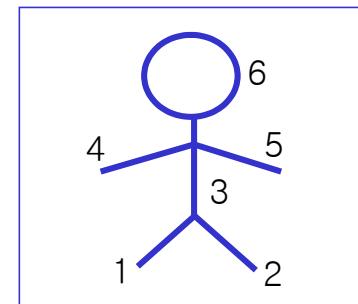
Frame Buffer (Computer Graphics 관점)

- **Buffer**: a region of memory used to store (temporary) data being processed.
- **Frame buffer**: stores data for graphics display
- 넓은 의미로 **프레임 버퍼**는 화면에 도시할 래스터 이미지뿐만 아니라 그러한 이미지를 생성하는데 필요한 여러 부류의 정보를 저장해주는 포괄적 의미의 그래픽스 전용 메모리를 뜻함.
- **Color buffer**: retain color values for pixels
 - **Double buffer**: used when moving objects are drawn
 - **Stereo buffer**: used for stereoscopic imaging
 - **Alpha buffer**: usually retain information on pixel transparency
- **Depth buffer (Z-buffer)**: retain a measure of the distance from the camera to the object point visible through each pixel
 - Used to implement a hidden surface removal algorithm (depth test).

- **Stencil buffer**
 - Usually used to limit the area of rendering
- **Framebuffer object (OpenGL)**
 - Used to do flexible off-screen rendering
 - 자체적으로 color buffer, depth buffer, stencil buffer를 구성할 수 있음.
- 기타

Double Buffering

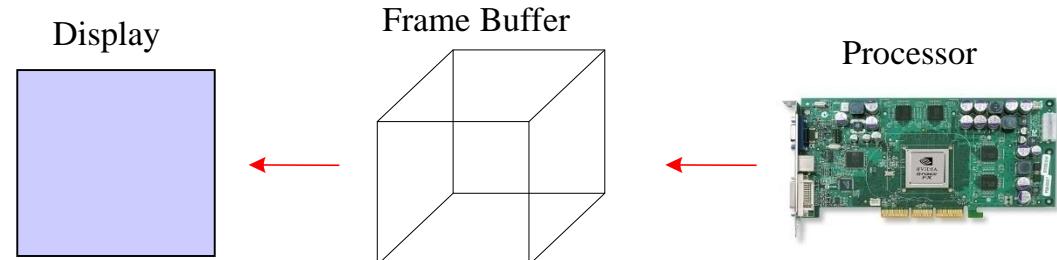
- 문제: 체조를 하고 있는 사람에 대한 애니메이션
- 방법:
 - 조금씩 움직이는 사람에 대하여 매 프레임을 계산하여 프레임 버퍼에 그림
 - 편의상 왼쪽 그림의 번호 순대로 계산을 한다고 가정.



Single buffering

- 프로세서는 새로운 이미지의 내용을 계산하여 프레임 버퍼에 차례대로 그림.
 - 동시에 비디오 제어기는 이와 무관하게 프레임 버퍼의 내용을 스캔하면서 화면에 도시.
- ★ 이미지가 생성되고 있는 도중의 내용이 예측할 수 없는 형태로 도시가 되기 때문에 불규칙한 이미지가 도시됨. → 매우 눈에 거슬리는 애니메이션 생성(유령 효과).

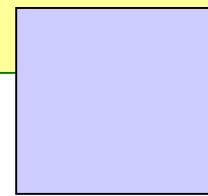
```
While (1) {  
    clear_buffer();  
    draw_next_frame();  
}
```



- Double buffering

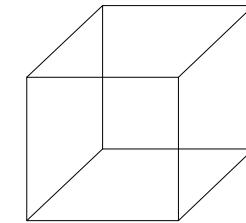
- 비디오 제어기가 항상 완성된 이미지를 도시하도록 함.
- 이를 위하여 두 개의 버퍼를 사용함
 - 프로세서는 이미지의 내용을 계산하여 뒤 버퍼에 축적.
 - 그 동안 비디오 제어기는 앞 버퍼의 내용을 읽어 화면에 이미지 도시.
- 싱글 버퍼링보다 훨씬 부드러운 애니메이션 생성.

```
glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE  
                     | GLUT_DEPTH);  
    // Allocate double buffer.  
  
while (1) {  
    glClear(GL_COLOR_BUFFER_BIT);  
        // Clear back buffer.  
    draw_next_frame(); // Into back buffer.  
    glutSwapBuffers(); // Swap buffers.  
}
```



Display

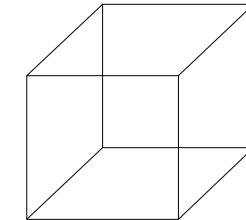
Frame Buffer
(Back)



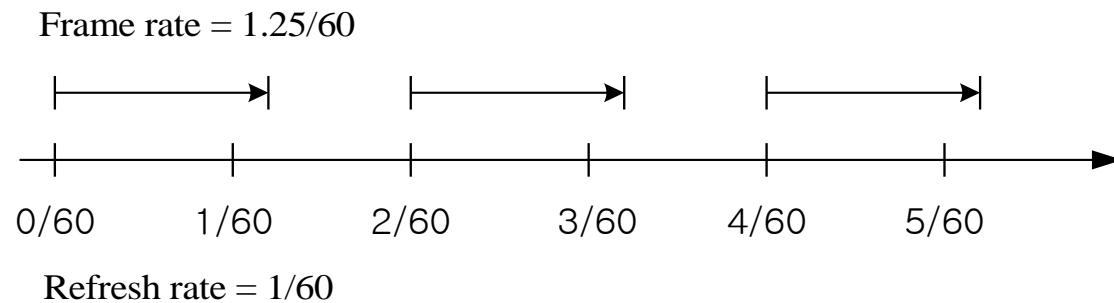
Processor



Frame Buffer
(Front)

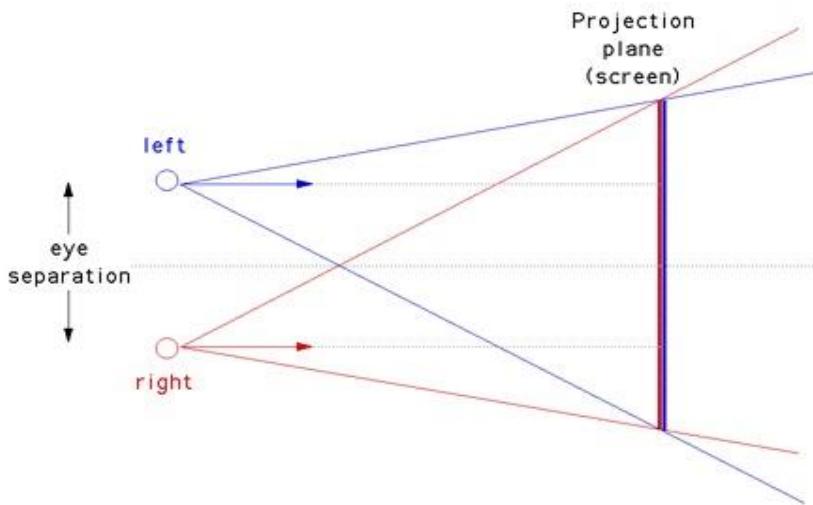


- 더블 버퍼링을 사용할 경우 고려할 점.
 - 버퍼가 하나 더 필요함. → 필요에 따라 주어진 색깔 버퍼를 두 개로 나누어야 함.
 - 실제 frame rate가 낮아질 수 있음.
 - 예: 60Hz → 30Hz → 20Hz → 15Hz → 12Hz → 10Hz → ...

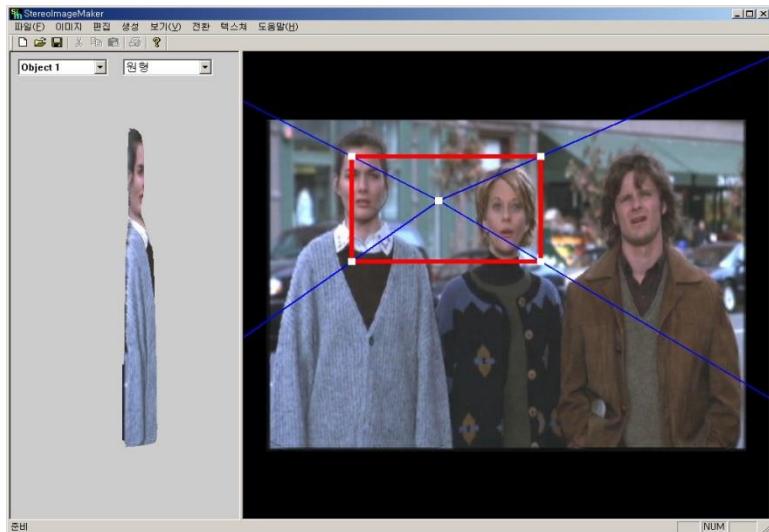


Stereoscopic Image

- 사람의 뇌는 약간의 각도 차이가 있는 왼쪽 눈과 오른쪽 눈이 인지한 두 이미지의 차이를 통하여 입체감, 즉 거리감을 느낀.
- 양쪽 눈에서 바라본 이미지 각각을 생성하여 저장하기 위하여 두 개의 버퍼, 즉 왼쪽 버퍼(left buffer)와 오른쪽 버퍼(right buffer)를 사용.
- 그래픽스 시스템이 두 이미지를 동시에 도시하면, 관찰자는 입체 안경 등을 통하여 입체감을 느낀.



- 입체 영상 활용 예



입체 영화 생성



입체 가시화 시스템



© 서강대학교 컴퓨터 그래픽스 연구실

EXERCISE 1: 프레임 버퍼에 대한 정보 출력하기

- 다음 OpenGL 함수의 목적과 사용법을 익힐 것.
 - `glGetBooleanv()`
 - `glGetDoublev()`
 - `glGetFloatv()`
 - `glGetIntegerv()`
 - 다음 GLUT 함수의 사용법을 익힐 것.
 - `glutInitDisplayMode()`
 - 수업에서 설명한 프로그램을 통하여 자신의 컴퓨터의 그래픽스 하드웨어가 지원하는 버퍼의 성능을 확인할 것.
- ✓ 디스플레이 등록 정보 → 설정 → 색 품질을 ‘중간(16비트)’과 ‘아주 높음(32비트)’ 등으로 바꾸어 가면서 실험해볼 것.

EXERCISE 2: PPM 형식에 대한 이해

- PPM = Portable Pixel Map
 - <http://netpbm.sourceforge.net/doc/ppm.html>을 읽고 PPM 포맷에 대하여 이해할 것.
 - PPM 형식으로 저장되어 있는 임의의 이미지를 읽어 'r', 'g', 'b' 키를 누를 때마다 각각 red, green, blue 채널의 이미지를 도시해주는 프로그램을 작성하라. 'o' 키를 누를 경우 원래의 이미지를 도시.

P3

```
# comments here
4 4
15
0 0 0 0 0 0 0 0 15 0 15
0 0 0 0 15 7 0 0 0 0 0
0 0 0 0 0 0 0 15 7 0 0 0
15 0 15 0 0 0 0 0 0 0 0 0
```

P6

```
# comments here
4 4
15
```

binary data

[CSE4170 기초 컴퓨터 그래픽스]

2019년도 1학기

강의자료 I

freeGLUT-based Window Programming

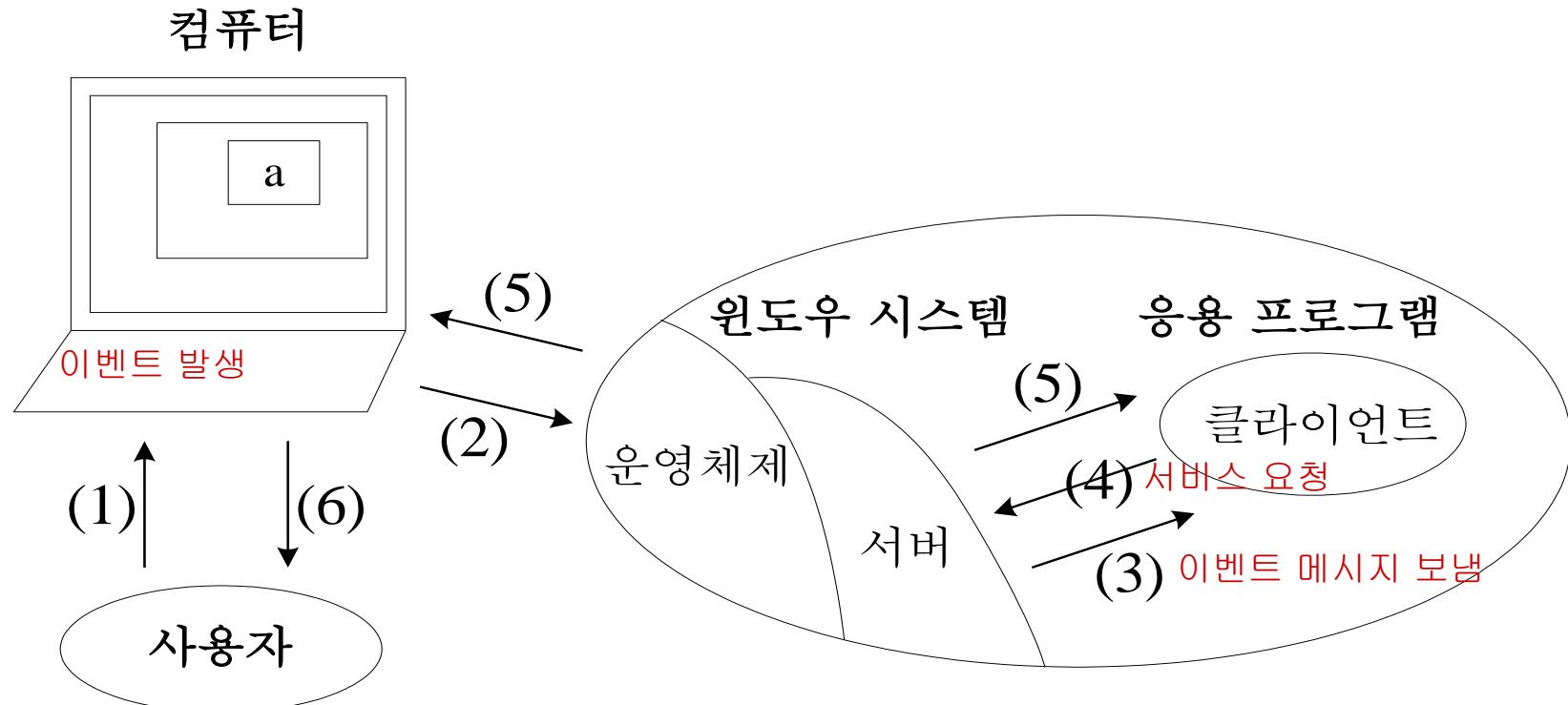
Window Programming for OpenGL

Window System

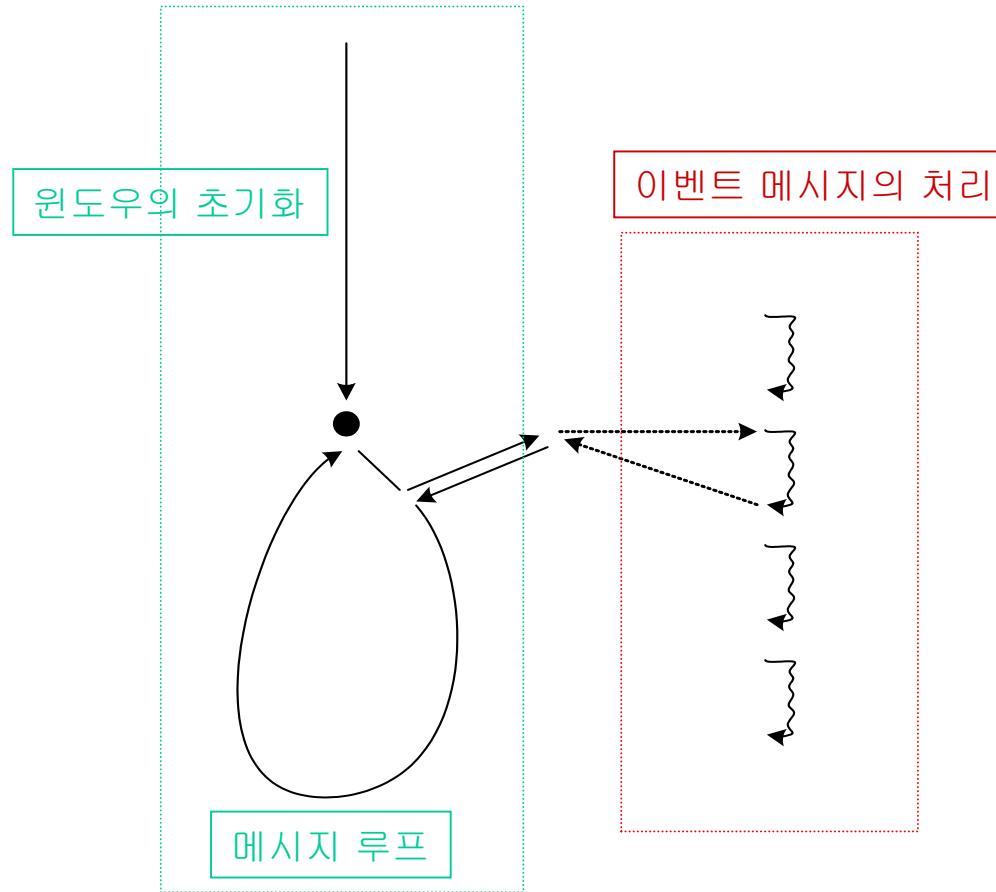
- 윈도우 시스템
 - Microsoft Windows
 - X Window System
- 윈도우 프로그래밍(window programming)
- 윈도우 시스템과 OpenGL 시스템은 모두 래스터 그래픽스 시스템임.
- OpenGL 프로그래밍을 하기 위해서는
 - ① 사용 윈도우 시스템에서 제공하는 래스터 시스템을 기반으로 윈도우 프로그래밍 수행
 - ② 윈도우 프로그래밍 문맥에서 추상적인 래스터 시스템인 OpenGL 시스템을 윈도우 시스템에 연결
 - ③ OpenGL에서 제공하는 함수들을 사용하여 3차원 그래픽스 프로그래밍을 수행
 - ④ 원하는 OpenGL 작업이 실제로 하드웨어를 제어하고 있는 사용 윈도우 시스템이 효율적으로 이해할 수 있는 형태로 전환
- ★ 각 윈도우 시스템은 효과적으로 OpenGL과 연결을 할 수 있도록 해주는 확장 함수 제공
 - 예: WGL, GLX, AGL, PGL, ...

Basic Concept of Window Programming

- 클라이언트-서버 시스템인 윈도우 시스템의 구조에 맞게 프로그래밍

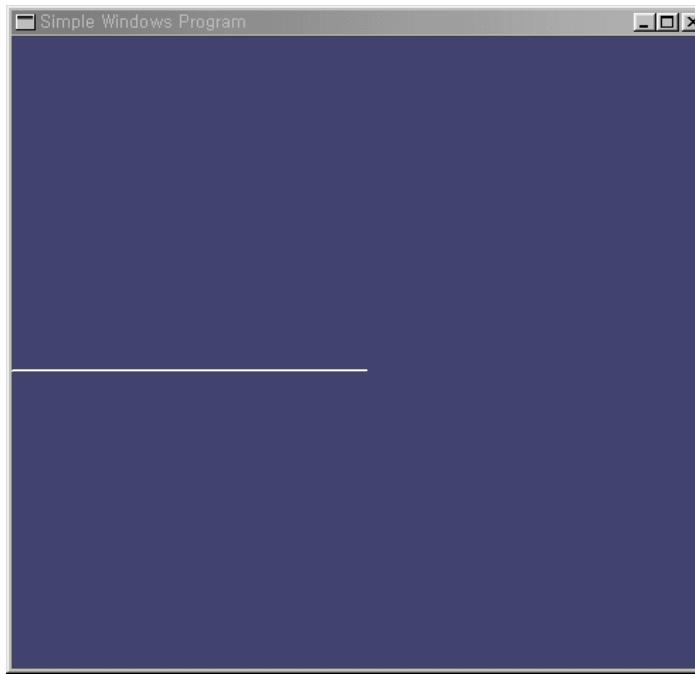


윈도우 응용 프로그램의 수행 구조



간단한 윈도우 프로그래밍의 예

- 윈도우스 프로그램 작성 예 (Win32)
- 윈도우스 프로그램과 OpenGL의 연결 예 (Win32 + WGL + OpenGL)
- GLUT를 사용한 OpenGL 프로그래밍 예 1 (GLUT + OpenGL)



윈도우스 프로그램 작성 예 (Win32)

- 윈도우 클래스 (window class)

```
typedef struct {  
    UINT cbSize;  
    UINT style;  
    WNDPROC lpfnWndProc;  
    int cbClsExtra;  
    int cbWndExtra;  
    HINSTANCE hInstance;  
    HICON hIcon;  
    HCURSOR hCursor;  
    HBRUSH hbrBackground;  
    LPCTSTR lpszMenuName;  
    LPCTSTR lpszClassName;  
    HICON hIconSm;  
} WNDCLASSEX;
```

- 윈도우 프로시저 (window procedure)
- 메시지 큐 (message queue)
- 메시지 루프 (message loop)
- 메시지 타입 (message type)

- MSG 구조

```
typedef struct {  
    HWND hwnd;  
    UINT message;  
    WPARAM wParam;  
    LPARAM lParam;  
    DWORD time;  
    POINT pt;  
} MSG;
```

- GDI (Graphics Device Interface)
- 디바이스 문맥 (Device Context, DC)

```

#include "stdafx.h"

LRESULT CALLBACK WndProc(HWND, UINT,
    WPARAM, LPARAM);

int APIENTRY WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPSTR lpCmdLine, int nCmdShow) {
    HWND hwnd; MSG msg; WNDCLASSEX wcex;
    static char szAppName[] = "SimpleWinProg";

    wcex.cbSize = sizeof(WNDCLASSEX);
    wcex.style = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc = (WNDPROC) WndProc;
    wcex.cbClsExtra = 0; wcex.cbWndExtra = 0;
    wcex.hInstance = hInstance;
    wcex.hIcon = LoadIcon(NULL, DI_APPLICATION);
    wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
    wcex.hbrBackground = (HBRUSH)
        CreateSolidBrush(RGB(66, 66, 111));
    wcex.lpszMenuName = NULL;
    wcex.lpszClassName = szAppName;
    wcex.hIconSm = LoadIcon(NULL,
       IDI_APPLICATION);
}

```

RegisterClassEx(&wcex);

```

hwnd = CreateWindow(szAppName,
    "SimpleWindows Program",
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT,
    500, 500, NULL, NULL, hInstance, NULL);

ShowWindow(hwnd, nCmdShow);
UpdateWindow(hwnd);

while (GetMessage(&msg, NULL, 0, 0)) {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}

return msg.wParam;
}

```

.

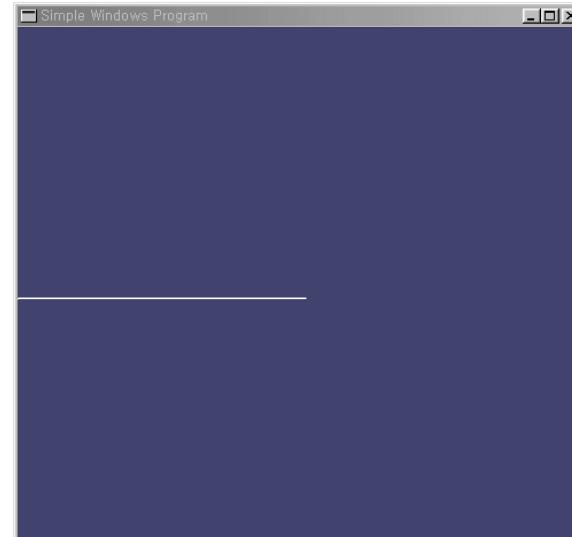
```
HRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam) {
    HDC hdc;
    PAINTSTRUCT ps;

    switch (message) {
        case WM_PAINT:
            hdc = BeginPaint(hwnd, &ps);
            SelectObject(hdc, GetStockObject(WHITE_PEN));
            MoveToEx(hdc, 0, 250, NULL);
            LineTo(hdc, 250, 250);
            EndPaint(hwnd, &ps);
            break;
        case WM_CHAR:
            if (wParam == 'q') {
                PostQuitMessage(0);
            }
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
    }
}
```

```
default:
    return DefWindowProc(hwnd, message, wParam, lParam);
}
```

윈도우스 프로그램과 OpenGL의 연결 예 (Win32 + WGL + OpenGL)

- GDI에 의해 정의되는 윈도우 시스템의 그래픽스 시스템과 OpenGL이 정의하는 그래픽스 시스템과의 연결 필요
 - ★ 사용 화소 형식(pixel format)에 대한 설정
 - ① OpenGL쪽에서 희망하는 프레임 버퍼의 성능을 화소 형식을 통하여 요청
 - ② 하드웨어에 대한 정보를 가지고 있는 GDI가 하드웨어가 지원을 하는 범위 내에서 OpenGL에서 요구한 성능을 최대로 만족시켜주는 화소 형식을 설정
 - ③ 윈도우 시스템의 그래픽스 작업에 관련된 모든 정보를 저장하는 디바이스 문맥에 저장을 한 후 사용



- PIXELFORMATDESCRIPTOR 구조

```
typedef struct tagPIXELFORMATDESCRIPTOR {  
    WORD    nSize;  
    WORD    nVersion;  
    DWORD   dwFlags;  
    BYTE    iPixelType;  
    BYTE    cColorBits;  
    BYTE    cRedBits;  
    BYTE    cRedShift;  
    BYTE    cGreenBits;  
    BYTE    cGreenShift;  
    BYTE    cBlueBits;  
    BYTE    cBlueShift;  
    BYTE    cAlphaBits;  
    BYTE    cAlphaShift;  
    BYTE    cAccumBits;  
    BYTE    cAccumRedBits;  
    BYTE    cAccumGreenBits;  
    BYTE    cAccumBlueBits;  
    BYTE    cAccumAlphaBits;  
    BYTE    cDepthBits;  
    BYTE    cStencilBits;  
    BYTE    cAuxBuffers;  
    BYTE    iLayerType;  
    BYTE    bReserved;  
    DWORD   dwLayerMask;  
    DWORD   dwVisibleMask;  
    DWORD   dwDamageMask;  
} PIXELFORMATDESCRIPTOR;
```

- PIXELFORMATDESCRIPTOR 구조 설정 예

```
PIXELFORMATDESCRIPTOR pfd = {
    sizeof(PIXELFORMATDESCRIPTOR), // size of this pfd
    1,                         // version number
    PFD_DRAW_TO_WINDOW |       // support window
    PFD_SUPPORT_OPENGL |       // support OpenGL
    PFD_DOUBLEBUFFER,          // double buffered
    PFD_TYPE_RGBA,             // RGBA type
    24,                        // 24-bit color depth
    0, 0, 0, 0, 0, 0,          // color bits ignored
    0,                         // no alpha buffer
    0,                         // shift bit ignored
    0,                         // no accumulation buffer
    0, 0, 0, 0,                // accum bits ignored
    32,                        // 32-bit z-buffer
    0,                         // no stencil buffer
    0,                         // no auxiliary buffer
    PFD_MAIN_PLANE,           // main layer
    0,                         // reserved
    0, 0, 0                    // layer masks ignored
};
```

- 렌더링 문맥 (Rendering Context, RC)

```

#include "stdafx.h"
#include <gl/gl.h>

LRESULT CALLBACK WndProc(HWND hwnd, UINT,
                        WPARAM,
                        LPARAM);
void SetDCPixelFormat(HDC hdc);

int APIENTRY WinMain(HINSTANCE hInstance,
                     ... , int nCmdShow) {
    ...
}

void SetDCPixelFormat(HDC hdc) {
    int nPixelFormat;

    static PIXELFORMATDESCRIPTOR pfd = {
        sizeof(PIXELFORMATDESCRIPTOR), 1,
        PFD_DRAW_TO_WINDOW |
        PFD_SUPPORT_OPENGL, PFD_TYPE_RGBA,
        24, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        PFD_MAIN_PLANE, 0, 0, 0, 0};

    nPixelFormat = ChoosePixelFormat(hdc, &pfd);
    SetPixelFormat(hdc, nPixelFormat, &pfd);
}

```

```

LRESULT CALLBACK WndProc(HWND hwnd,
                        UINT message, WPARAM
                        wParam,
                        LPARAM lParam) {
    HDC hdc; HGLRC hrc;

    switch (message) {
        case WM_CREATE:
            hdc = GetDC(hwnd);
            SetDCPixelFormat(hdc);
            hrc = wglCreateContext(hdc);
            wglMakeCurrent(hdc, hrc);
            break;
        case WM_PAINT:
            glClearColor(0.259, 0.259, 0.435, 1.0);
            glClear(GL_COLOR_BUFFER_BIT);
            glColor3f(1.0, 1.0, 1.0);
            glBegin(GL_LINES);
            glVertex2f(-1.0, 0.0);
            glVertex2f(0.0, 0.0);
            glEnd();
            glFlush();
            ValidateRect(hwnd, NULL);
            break; WW
    }
}

```

```
case WM_CHAR:  
    if (wParam == 'q') {  
        wglGetCurrent(hdc, NULL);  
        wglDeleteContext(hrc);  
        PostQuitMessage(0);  
    }  
    break;  
case WM_DESTROY:  
    wglGetCurrent(hdc, NULL);  
    wglDeleteContext(hrc);  
    PostQuitMessage(0);  
    break;  
default:  
    return DefWindowProc(hwnd, message,  
    wParam,  
    lParam);  
}  
return 0;  
}
```



```
case WM_CHAR:  
    if (wParam == 'q') {  
        if (hrc = wglGetCurrentContext() ) {  
            hdc = wglGetCurrentDC();  
            wglMakeCurrent(NULL, NULL);  
            ReleaseDC (hwnd, hdc);  
            wglDeleteContext(hrc);  
        }  
        PostQuitMessage(0);  
    }  
    break;  
case WM_DESTROY:  
    break;  
default:  
    return DefWindowProc(hwnd, message, wParam,  
    lParam);  
}  
return 0;  
}
```

wglMakeCurrent

The **wglMakeCurrent** function makes a specified OpenGL rendering context the calling thread's current rendering context. All subsequent OpenGL calls made by the thread are drawn on the device identified by *hdc*. You can also use **wglMakeCurrent** to change the calling thread's current rendering context so it's no longer current.

```
BOOL wglMakeCurrent( HDC hdc, // device context of device that OpenGL calls are // to be drawn on HGLRC hglrc //  
OpenGL rendering context to be made the calling // thread's current rendering context );
```

Parameters

hdc

Handle to a device context. Subsequent OpenGL calls made by the calling thread are drawn on the device identified by *hdc*.

hglrc

Handle to an OpenGL rendering context that the function sets as the calling thread's rendering context. If *hglrc* is NULL, the function makes the calling thread's current rendering context no longer current, and releases the device context that is used by the rendering context. In this case, *hdc* is ignored.

Return Values

When the **wglMakeCurrent** function succeeds, the return value is TRUE; otherwise the return value is FALSE. To get extended error information, call **GetLastError**.

Window Programming Using freeGLUT

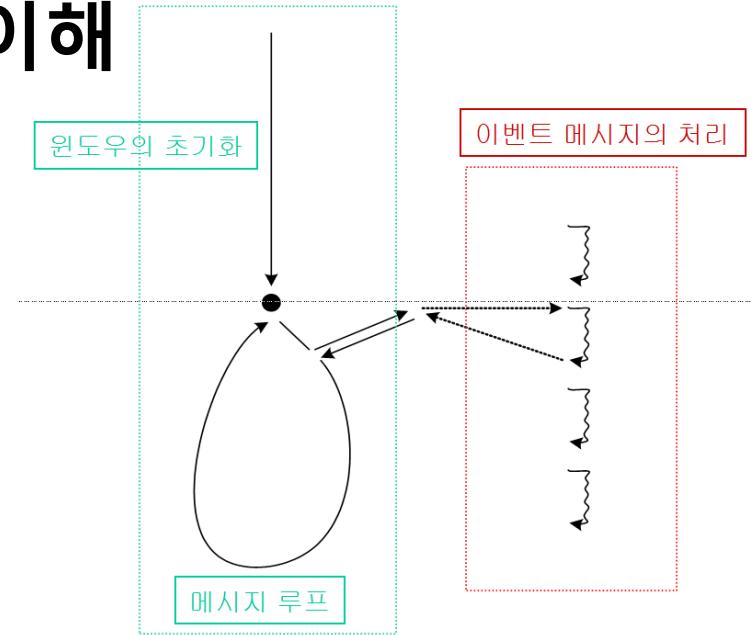
GLUT and freeGLUT

- 주어진 윈도우 시스템을 기반으로 한 OpenGL (ES) 프로그래밍
 - ① 윈도우 시스템에서 제공하는 기본적인 윈도우 함수들을 사용하여 윈도우 시스템과 OpenGL 시스템을 연결함.
 - ② 일반적인 윈도우 프로그래밍은 윈도우 시스템에서 사용하는 윈도우 함수를 사용함.
 - ③ 3D 그래픽스와 관련한 프로그래밍은 OpenGL (ES) 함수들을 사용하여 수행함.
- GLUT(GLUT Utility Toolkit) v3.7
 - PC의 Windows 환경이 제공하는 low-level 함수를 사용할 필요가 없이, high-level 함수를 사용하여 윈도우 프로그래밍에 관련된 작업을 손쉽게 수행할 수 있도록 해주는 툴킷
 - 편리한 대신 세밀한 윈도우 프로그래밍 기능은 사용할 수 없음.
 - 참고 자료
 - ★ <http://www.opengl.org/resources/libraries/glut.html>
 - ★ <http://www.opengl.org/resources/libraries/glut/spec3/spec3.html>
 - ★ <http://www.opengl.org/resources/libraries/glut/glutdlls37beta.zip>
 - ★ <http://www.cs.csustan.edu/%7ersc/SDSU/GLUTinstall.html>

- **freeGLUT (v3.0.0)**
 - A **free-software/open-source alternative to the OpenGL Utility Toolkit (GLUT) library**
 - GLUT는 지금도 사용이 가능하나 더 이상 3.7 버전 이후로 유지 관리가 되고 있지 않음.
 - 본 과목에서는 GLUT의 기능을 개선 확장한 freeGLUT를 사용하여 PC 환경에서의 윈도우 프로그래밍을 대치함.
 - 참고 자료
 - ★ <http://freeglut.sourceforge.net/>
 - ★ <http://freeglut.sourceforge.net/docs/api.php>
 - 주의
 - 기본적으로 GLUT와 관련한 자료를 본 후, freeGLUT에서 확장한 부분을 이해해야 함.
 - 설치 방법은 조교가 설명할 예정임.
 - 윈도우 프로그래밍에 관심이 있는 학생은 open-source SW인 이 툴킷의 원시 코드를 살펴보기 바람.

예제를 통한 freeGLUT 프로그래밍 이해

```
void main(int argc, char *argv[]) {  
    glutInit(&argc, argv);  
    glutInitContextVersion(4, 0);  
    glutInitContextProfile(GLUT_COMPATIBILITY_PROFILE);  
  
    glutInitDisplayMode(GLUT_RGBA);  
  
    glutInitWindowSize(500, 500);  
    glutInitWindowPosition(500, 300);  
    glutCreateWindow("My first OpenGL program");  
  
    initialize_glew();  
    register_callbacks();  
    glutSetOption(GLUT_ACTION_ON_WINDOW_CLOSE,  
                 GLUT_ACTION_GLUTMAINLOOP_RETURNS);  
    glutMainLoop();  
}
```



```
void register_callbacks(void) {  
    glutDisplayFunc(display);  
    glutKeyboardFunc(keyboard);  
    glutSpecialFunc(special);  
    glutMouseFunc(mousepress);  
    glutMotionFunc(mousemove);  
    glutReshapeFunc(reshape);  
    glutCloseFunc(close);  
}
```

freeGLUT 주요 함수

- freeGLUT 함수를 이용한 윈도우에 대한 초기화

- `void glutInit(int *argcp, char **argv); // Initialize the GLUT library.`
- `void glutInitDisplayMode(unsigned int mode); // Set the initial display mode.`

Buffer	GLUT BIT MASK	Purpose
Color Buf.	GLUT_RGBA	
	GLUT_RGB	
	GLUT_INDEX	
	GLUT_ALPHA	
	GLUT_SINGLE	
	GLUT_DOUBLE	
	GLUT_STEREO	
	GLUT_LUMINANCE	
	GLUT_MULTISAMPLING	
Other Buf.'s	GLUT_ACCUM	
	GLUT_DEPTH	
	GLUT_STENCIL	

- `void glutInitWindowSize(int width, int height);` // Set the initial window size.
- `void glutInitWindowPosition(int x, int y);` // Set the initial window position.
- `int glutCreateWindow(char *name);` // Create top-level window.
 - Each created window has a unique associated OpenGL context.
 - State changes to a window's associated OpenGL context can be done immediately after the window is created.
 - The display state of a window is initially for the window to be shown. But the window's display state is not actually acted upon until `glutMainLoop` is entered.
 - This means until `glutMainLoop` is called, rendering to a created window is ineffective because the window can not yet be displayed.
 - The value returned is a unique small integer identifier for the window. The range of allocated identifiers starts at one. This window identifier can be used when calling `glutSetWindow`.

- `void glutSetOption(GLenum eWhat, int value); // Set some general state/option variables.`
 - `eWhat`
 - `GLUT_ACTION_ON_WINDOW_CLOSE` (controls what happens when a window is closed by the user or system)
 - `value`
 - `GLUT_ACTION_EXIT` (immediately exit the application – default GLUT's behavior).
 - `GLUT_ACTION_GLUTMAINLOOP_RETURNS` (immediately return from the main loop)
 - `GLUT_ACTION_CONTINUE_EXECUTION` (continue execution of remaining windows)
- `void glutMainLoop(void); // Enter the GLUT event processing loop.`

- 이벤트 메시지의 처리를 위한 컬백 함수 등록
 - 컬백 함수 (**callback function**)
 - 윈도우 프로그램 수행 시 발생하는 여러 종류의 윈도우 및 입력 이벤트 중 관심이 있는 이벤트 각각에 대하여 어떠한 처리를 하고자 한다는 것을 기술해주는 함수
 - ① 프로그래머가 관심이 있는 이벤트 종류 각각에 대하여 컬백 함수를 정의한 후 등록(registration)
 - ② 서버로부터 해당 이벤트 메시지가 들어왔을 때 GLUT가 어떤 이벤트 메시지인지 를 파악한 후 그것에 등록되어 있는 컬백 함수를 호출
- `void glutDisplayFunc(void (*func) (void)); // Set the display callback for the current window.`
- `void glutKeyboardFunc(void (*func) (unsigned char key, int x, int y)); // Set the keyboard callback for the current window.`
- `void glutSpecialFunc(void (*func) (int key, int x, int y)); // Set the special keyboard callback for the current window.`

- `void glutMouseFunc(void (*func)(int button, int state, int x, int y));` // Set the **mouse callback** for the current window.
 - button: GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, or GLUT_RIGHT_BUTTON
 - state: GLUT_UP or GLUT_DOWN
- `void glutMotionFunc(void (*func)(int x, int y));` // Set the **motion callback** for the current window.
- `void glutReshapeFunc(void (*func)(int width, int height));` // Set the **reshape callback** for the current window.
 - The reshape callback is triggered when a window is reshaped.
 - A reshape callback is also triggered **immediately before a window's first display callback** after a window is created.
 - 이 시점은 `glutCreateWindow` 함수에 의해 내부적으로 윈도우 객체가 생성되는 시점과 함께 프로그램 수행에 필요한 초기화를 위한 시점을 제공함.

- `void glutCloseFunc(void (*func) (void));` // Set the **close callback** for the current window.
 - Specifies the function that freeglut will call to notify the application that a window is about to be closed.

- **두 유용한 freeGLUT 함수**

- `void glutPostRedisplay(void);` // Set the **mouse callback** for the current window.
 - Mark the normal plane of current window as needing to be redisplayed.
 - The next iteration through `glutMainLoop`, the window's display callback will be called to redisplay the window's normal plane.
 - Multiple calls to `glutPostRedisplay` before the next display callback opportunity generates only a single redisplay callback.
- `void glutLeaveMainLoop(void);` // Causes freeglut to stop the event loop.
 - If the `GLUT_ACTION_ON_WINDOW_CLOSE` option has been set to `GLUT_ACTION_GLUTMAINLOOP_RETURNS` or `GLUT_ACTION_CONTINUE_EXECUTION`, control will return to the function which called `glutMainLoop`;
 - Otherwise the application will exit.

```

#include <stdio.h>
#include <stdlib.h>
#include <GL/glew.h>
#include <GL/freeglut.h>

int rightbuttonpressed = 0;
float r = 0.0f, g = 0.0f, b = 0.0f;

void display(void) {
    glClearColor(r, g, b, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(1.0f, 1.0f, 1.0f);
    glBegin(GL_LINES);
    glVertex2f(-1.0f, 0.0f); glVertex2f(0.0f, 0.0f);
    glEnd();

    glFlush();
}

```

```

void keyboard(unsigned char key, int x, int y) {
    switch (key) {
        case 'r':
            r = 1.0f; g = b = 0.0f;
            glutPostRedisplay();
            break;
        case 'g':
            g = 1.0f; r = b = 0.0f;
            glutPostRedisplay();
            break;
        case 'b':
            b = 1.0f; r = g = 0.0f;
            glutPostRedisplay();
            break;
        case 'q':
            glutLeaveMainLoop();
            break;
    }
}

```

```

void special(int key, int x, int y) {
    switch (key) {
        case GLUT_KEY_LEFT:
            r -= 0.1f; if (r < 0.0f) r = 0.0f;
            glutPostRedisplay();
            break;
        case GLUT_KEY_RIGHT:
            r += 0.1f; if (r > 1.0f) r = 1.0f;
            glutPostRedisplay();
            break;
        case GLUT_KEY_DOWN:
            g -= 0.1f; if (g < 0.0f) g = 0.0f;
            glutPostRedisplay();
            break;
        case GLUT_KEY_UP:
            g += 0.1f; if (g > 1.0f) g = 1.0f;
            glutPostRedisplay();
            break;
    }
}

```

```

void mousepress(int button, int state, int x, int y) {
    if ((button == GLUT_LEFT_BUTTON) &&
        (state == GLUT_DOWN))
        fprintf(stdout, "*** The left mouse button was
                           pressed at (%d, %d).%n", x, y);
    else if ((button == GLUT_RIGHT_BUTTON) &&
              (state == GLUT_DOWN))
        rightbuttonpressed = 1;
    else if ((button == GLUT_RIGHT_BUTTON) &&
              (state == GLUT_UP))
        rightbuttonpressed = 0;
}

void mousemove(int x, int y) {
    if (rightbuttonpressed)
        fprintf(stdout, $$$ The right mouse button is
                           now at (%d, %d).%n", x, y);
}

void reshape(int width, int height) {
    fprintf(stdout, ### The new window size is
                           %dx%d.%n", width, height);
}

void close(void) {
    fprintf(stdout, "%n^^^ The control is at the close
                           callback function now.%n%n");
}

```