



**KOM1304 Grafika Komputer
dan Visualisasi**

Pengantar ke Grafika Komputer dan Visualisasi

**Auzi Asfarian, M Asyhar Agmalaro,
Endang P Giri, Shervie N Neyman**

Departemen Ilmu Komputer, FMIPA
IPB University (Institut Pertanian Bogor), Bogor, Indonesia
asfarian@apps.ipb.ac.id | ux@apps.ipb.ac.id



IPB University
— Bogor Indonesia —

Departemen
Ilmu Komputer

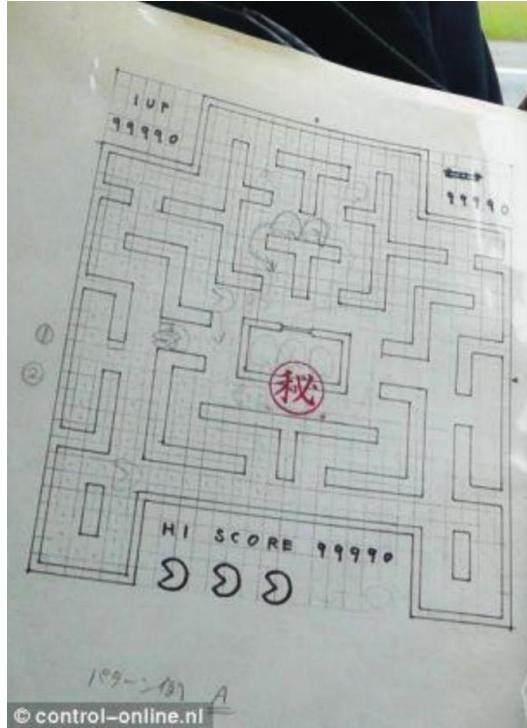
Bagian 1:

GKV Early Days vs Current Dyas

“Grafika Komputer” dan “Visualisasi”

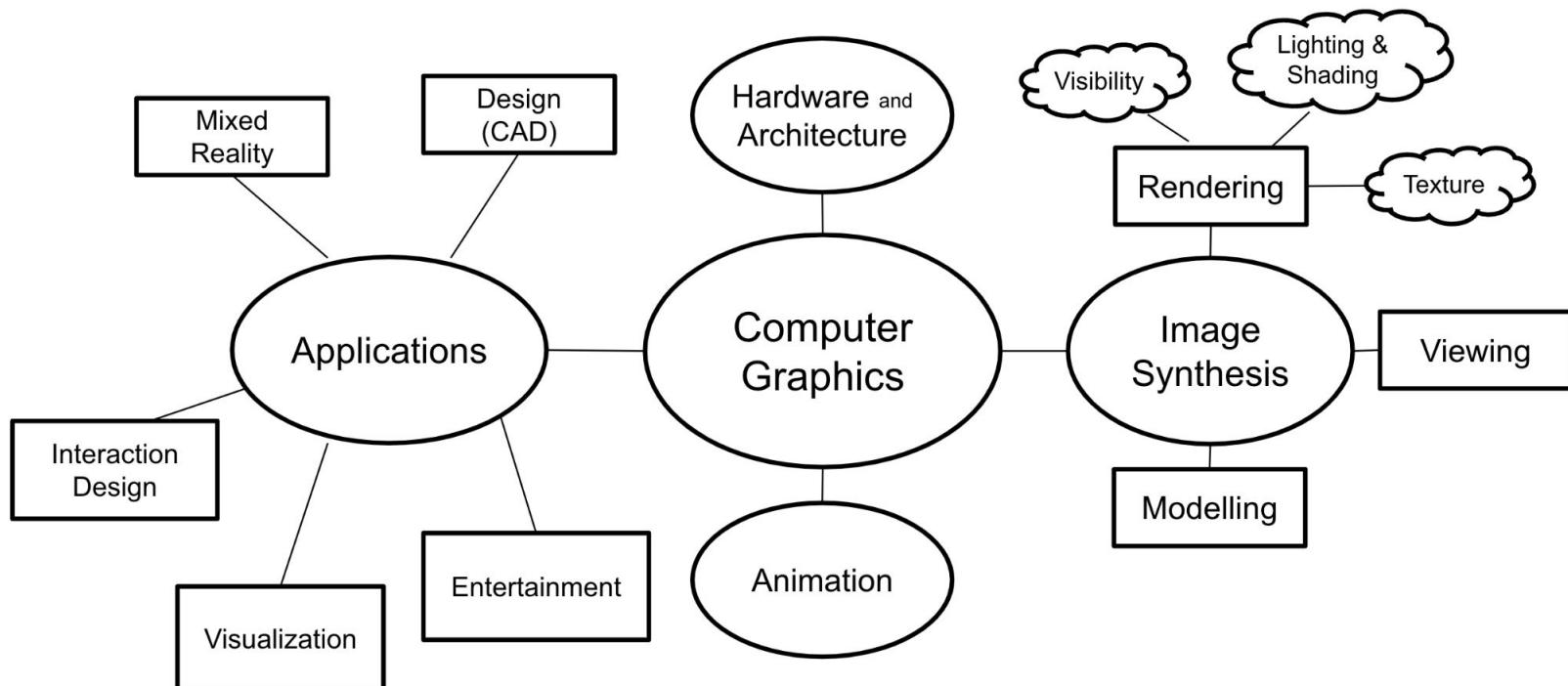


“Grafika Komputer” dan “Visualisasi”



dailymail.co.uk/sciencetech/article-1289239/Japanese-inventor-Pac-Man-reveals-original-sketches-iconic-video-game.html

“Grafika Komputer” dan “Visualisasi”

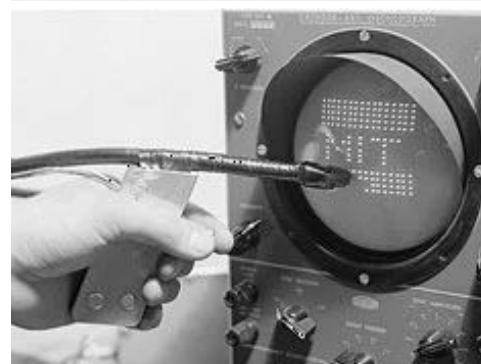


Diadopsi dari Torsten Möller

GKV “Early Days”



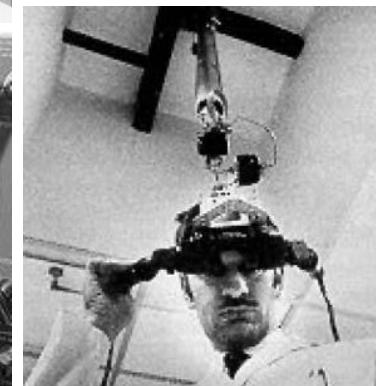
Laposky BF. 1953. Oscillons: Electronic Abstractions.



The Whirlwind computer at the Massachusetts Institute of Technology was the first computer with a video display of real time data.



Shuttleworth IE. 1963. Sketchpad, a man-machine graphical communication system (Disertasi). Massachusetts (US): MIT.



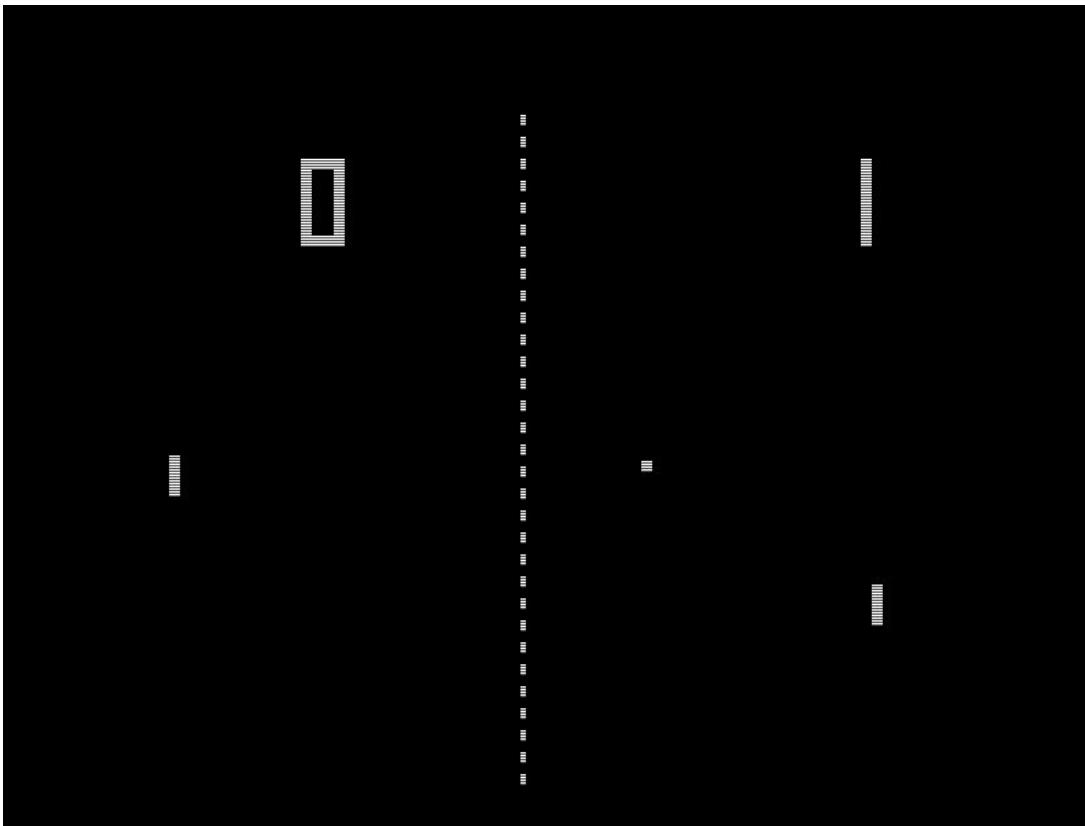
Shuttleworth IE. 1965. The Ultimate Display. Proceedings of IFIPS Congress. New York, Mei 1965. Hal. 506-508.



Feiner S, MacIntyre B, Höllerer T, Webster A. A touring machine: Prototyping 3D mobile augmented reality systems for exploring the urban environment. Personal Technologies. 1997 Dec 1;1(4):208-17.



Higinbotham W. Tennis for two. Analog computer/oscilloscope game, USA:
Brookhaven National Laboratory. 1958.



Pong yang dibuat oleh Atari pada tahun 1972.

THE NEWEST 2 PLAYER
VIDEO SKILL GAME

PONG

from ATARI CORPORATION
SYZYGY ENGINEERED

The Team That Pioneered Video Technology

FEATURES

- STRIKING Attract Mode
- Ball Serves Automatically
- Realistic Sounds of Ball Bouncing, Striking Paddle
- Simple to Operate Controls
- ALL SOLID STATE TV and Components for Long, Rugged Life
- ONE YEAR COMPUTER WARRANTY
- Prove HIGH PROFITS in Location After Location
- Low Key Cabinet, Suitable for Sophisticated Locations
- 25¢ per play

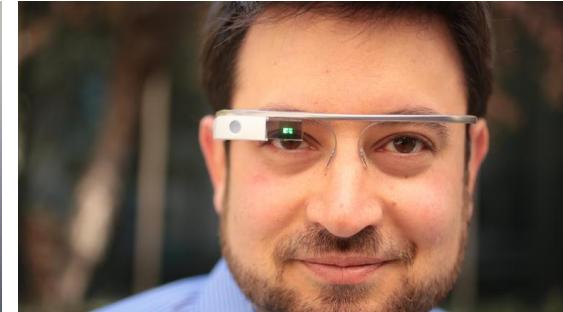
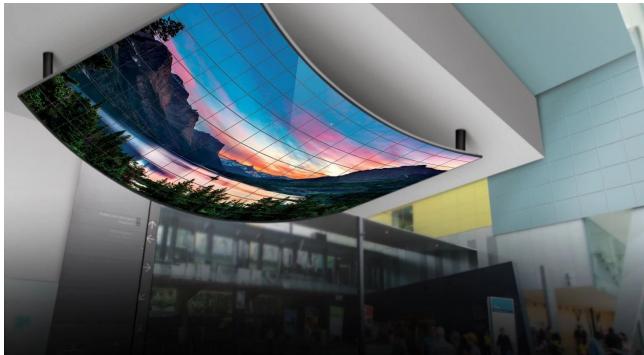
THIS GAME IS AVAILABLE FROM YOUR LOCAL DISTRIBUTOR

Manufactured by
ATARI, INC.
2962 SCOTT BLVD.
SANTA CLARA, CA.
95050

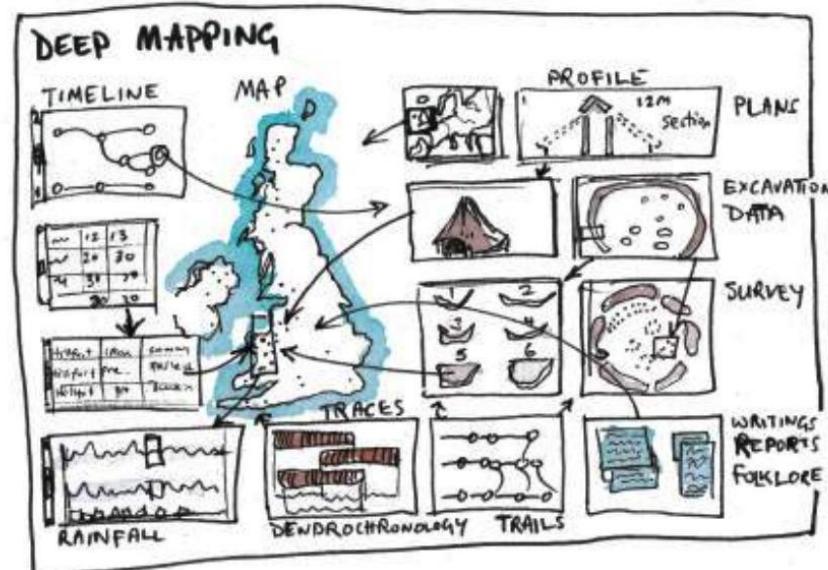
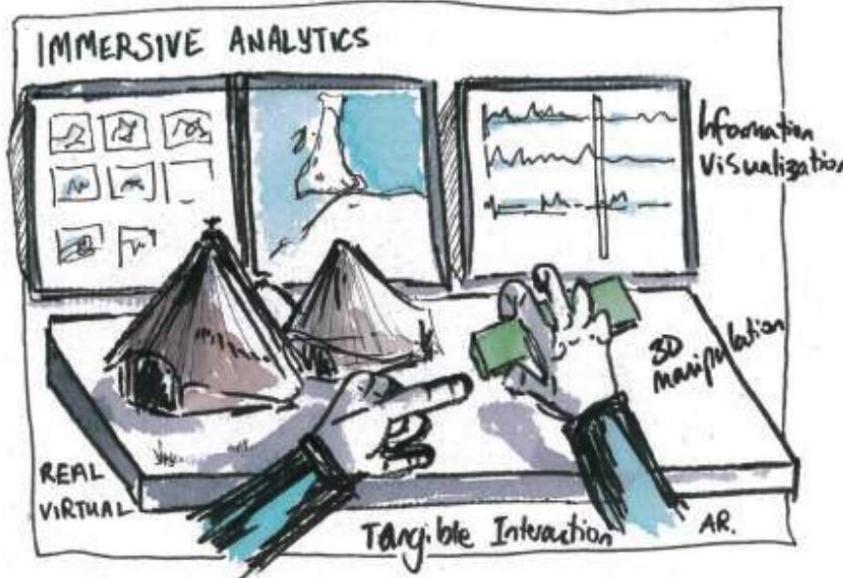
Maximum Dimensions:
WIDTH - 26"
HEIGHT - 50"
DEPTH - 24"
SHIPPING WEIGHT:
150 Lb.

Lowood H. Videogames in computer space: The complex history of pong. IEEE Annals of the History of Computing. 2009 Aug 25;31(3):5-19.

GKV “Current Days”



GKV “Current Days”



Roberts JC, Mearman JW, Ritsos PD, Miles HC, Wilson AT, Perkins DE, Jackson JR, Tiddeman B, Labrosse F, Edwards B, Karl R.
Position Paper: Immersive Analytics and Deep Maps—the Next Big Thing for Cultural Heritage & Archaeology.



GKV “Current Days”

SIGGRAPH 2021: Technical Papers Preview Trailer
youtube.com/watch?v=Ros7ZXqLbFg

“Grafika Komputer” dan “Visualisasi”

“Computer Graphics is the science and art of communicating visually via a computer’s display and its interaction devices.”

Computer Graphics: Principles and Practice.
Hughes et al. (2014)

“Graphics and Visualization: Design and develop a user interface using a standard API and that incorporates visual and audio techniques used for a local organization”

Computing Curricula 2020 Competency Draft
ACM and IEEE (2021)



IPB University
— Bogor Indonesia —

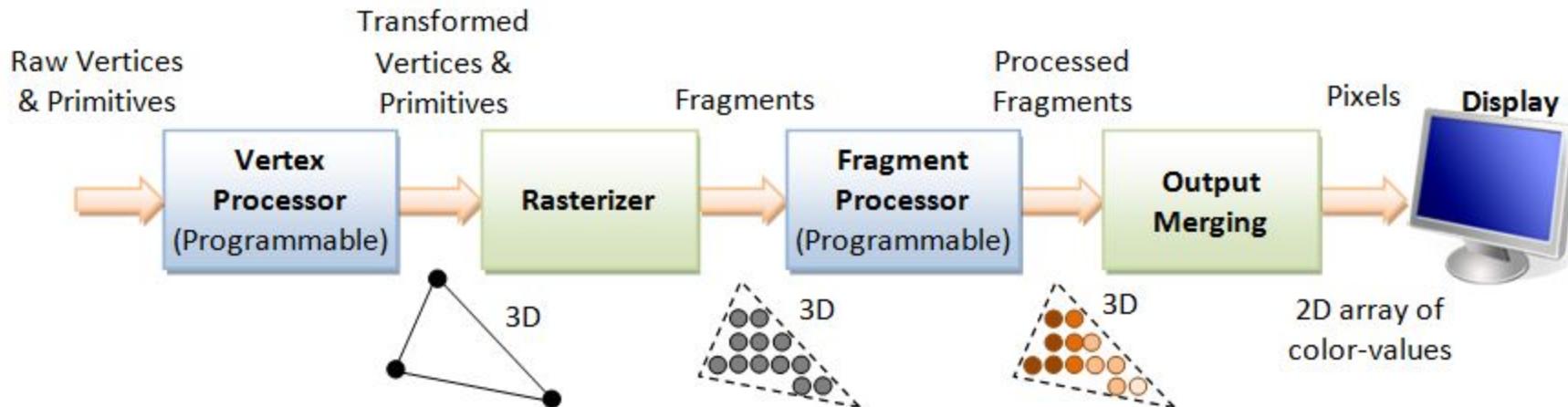
Departemen
Ilmu Komputer

Bagian 2: **Graphics API**

Kita Perlu Bicara Lebih Teknikal Tentang Ini



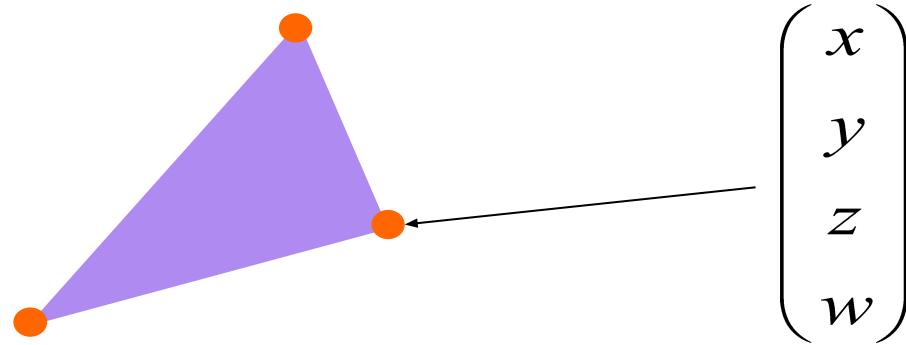
Graphics Pipeline



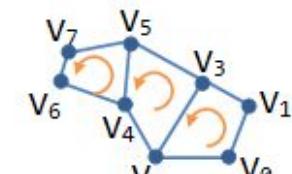
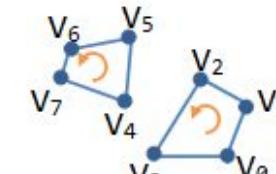
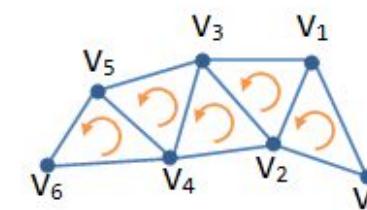
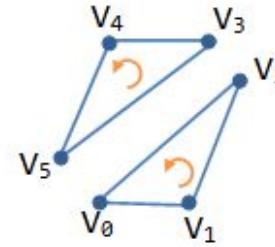
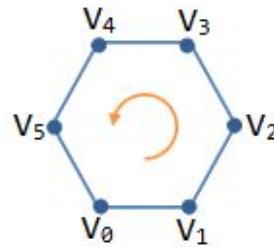
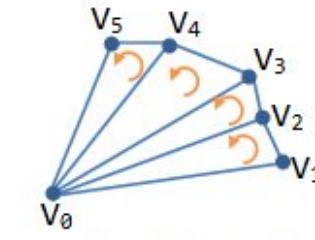
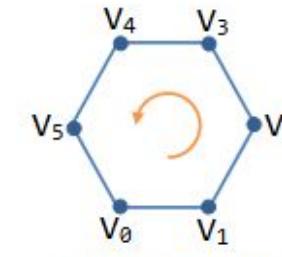
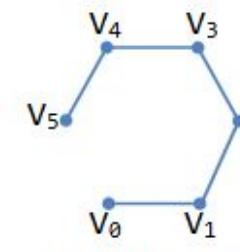
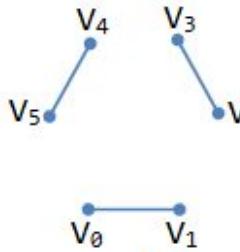
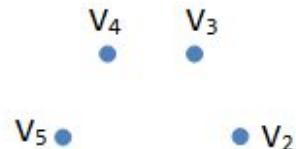
3D Graphics Rendering Pipeline: Output of one stage is fed as input of the next stage. A vertex has attributes such as (x, y, z) position, color (RGB or RGBA), vertex-normal (n_x, n_y, n_z) , and texture. A primitive is made up of one or more vertices. The rasterizer raster-scans each primitive to produce a set of grid-aligned fragments, by interpolating the vertices.

Sumber: **3D Graphics with OpenGL**
www.ntu.edu.sg

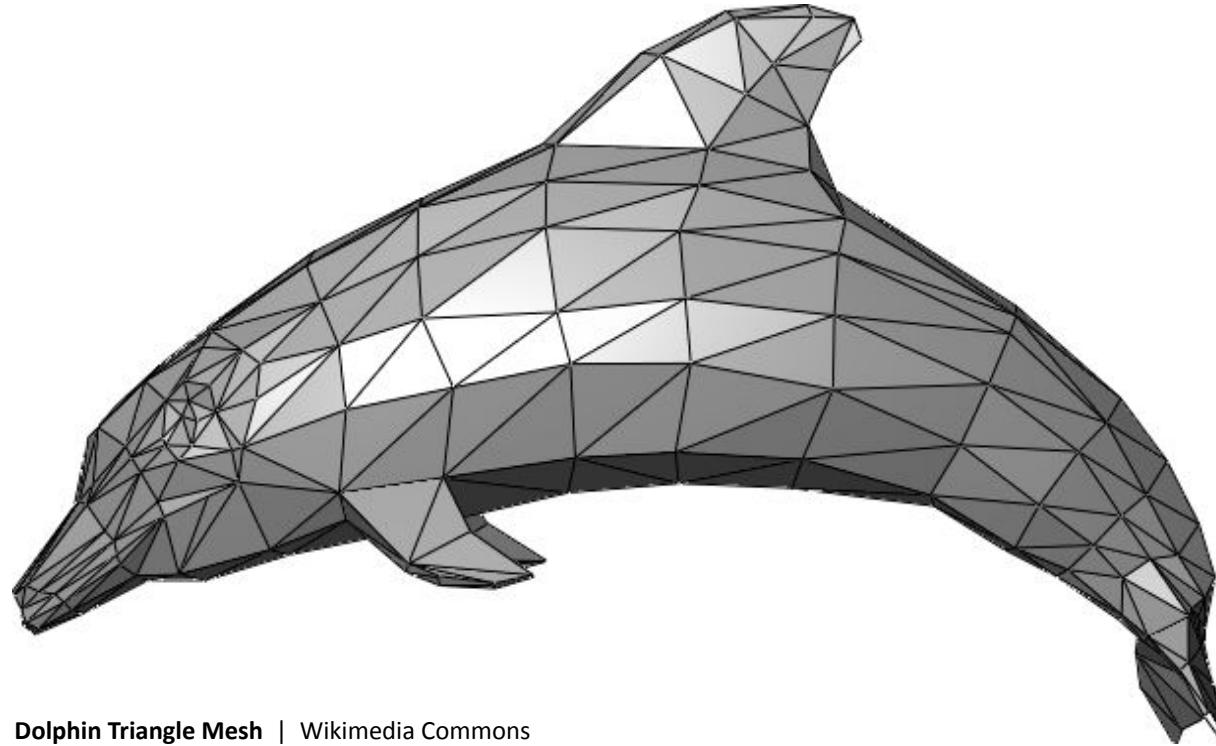
Verteks



Representasi Objek Geometri

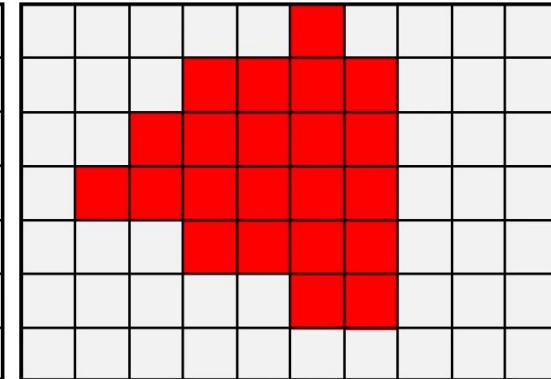
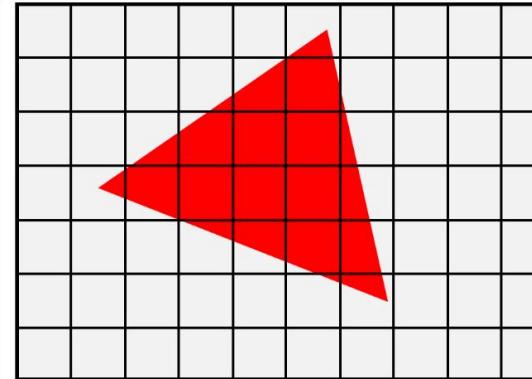
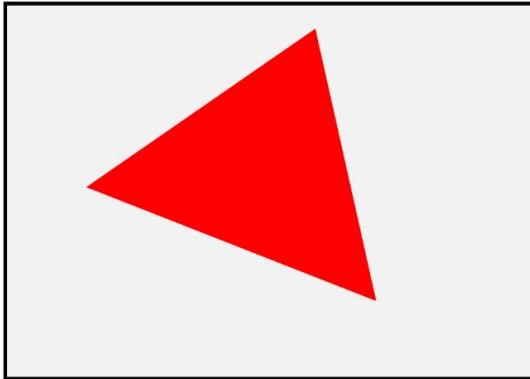


Representasi Objek Geometri

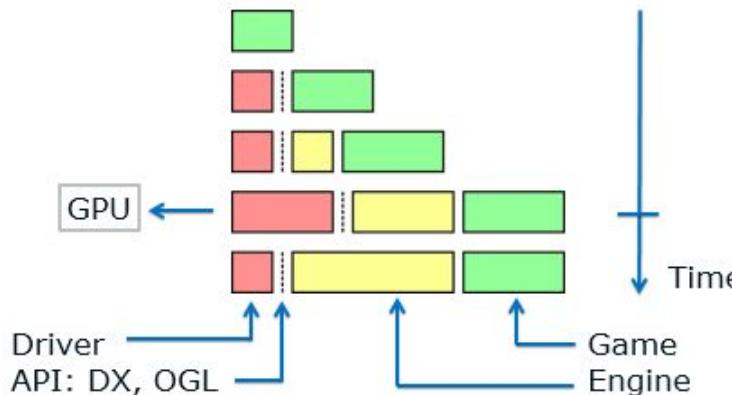


Dolphin Triangle Mesh | Wikimedia Commons

Rasterisasi



Graphic API



Application programming interface (API) adalah **kumpulan fungsi, protokol, dan alat** untuk mengembangkan perangkat lunak.

Tujuannya adalah untuk **mempermudah** dan **mempercepat** proses pengembangan perangkat lunak.

Salah satunya -> **API untuk grafika komputer.**

Graphic API

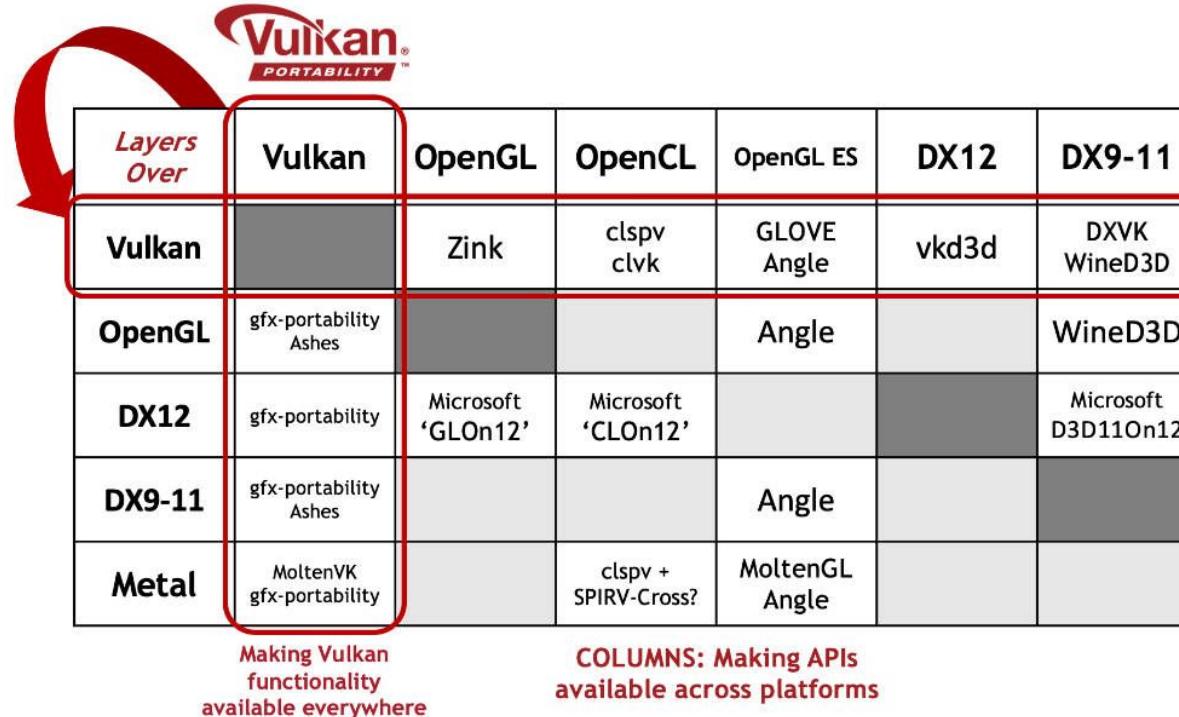
Low-Level API

- OpenGL
- OpenGL ES (mobile)
- Processing
- Direct3D (Microsoft)
- RenderMan
- Glide
- X3D
- WebGL (web-based)
- Stage3D (web-based)
- Vulcan
- Metal

High-Level API

- Java 3D
- Horde3D
- CrystalSpace
- OpenGL Performer
- OpenSceneGraph
- QSDK
- CopperLicht
- PaperVision3D (web-based)
- Unity
- Unreal Engine

Hubungan di Antara Graphic API (Khronos-Based)



<i>Layers Over</i>	Vulkan	OpenGL	OpenCL	OpenGL ES	DX12	DX9-11
Vulkan		Zink	clspv clvk	GLOVE Angle	vkd3d	DXVK WineD3D
OpenGL	gfx-portability Ashes			Angle		WineD3D
DX12	gfx-portability	Microsoft 'GLOn12'	Microsoft 'CLOn12'			Microsoft D3D11On12
DX9-11	gfx-portability Ashes			Angle		
Metal	MoltenVK gfx-portability		clspv + SPIRV-Cross?	MoltenGL Angle		

Making Vulkan functionality available everywhere

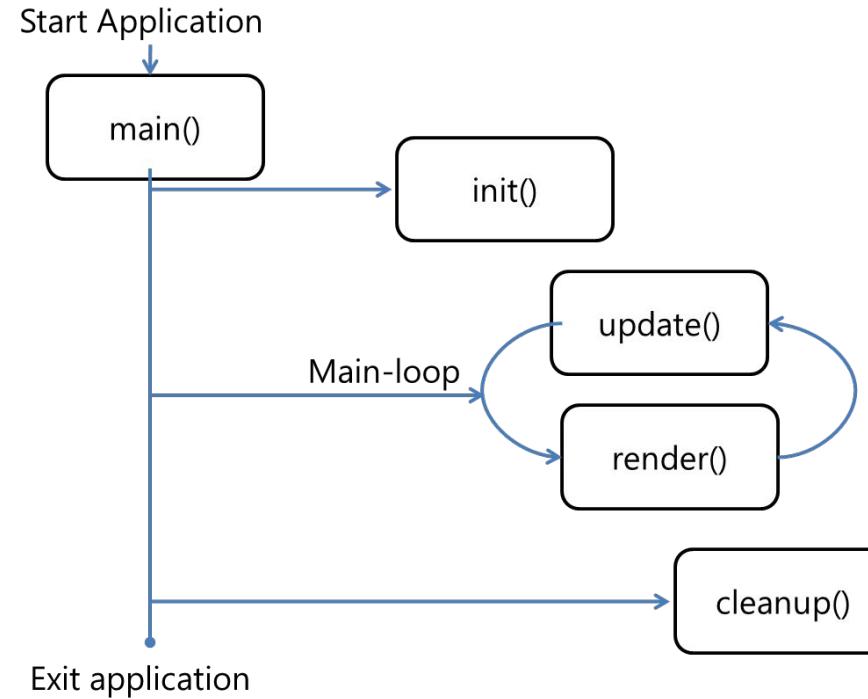
COLUMNS: Making APIs available across platforms

Vulkan is an effective porting target for multiple APIs

ROWS:
Bring more APIs to a Platform

Fighting Fragmentation: Vulkan Portability Extension Released and Implementations Shipping.
khronos.org/blog/fighting-fragmentation-vulkan-portability-extension-released-implementations-shipping

Struktur Umum Aplikasi Grafis





IPB University
— Bogor Indonesia —

Departemen
Ilmu Komputer

Bagian 3: Canvas API

Canvas API



Canvas API menyediakan sarana untuk menggambar grafik melalui JavaScript dan elemen <canvas> HTML.

Dapat digunakan untuk animasi, grafik game, visualisasi data, manipulasi foto, dan pemrosesan video waktu nyata.

Canvas API sebagian besar berfokus pada grafik 2D. Untuk grafik 3D menggunakan WebGL API.

Canvas API

```
1: <html>
2:   <canvas id="canvas"></canvas>
3: </html>
4: <script>
5:   const canvas = document.getElementById('canvas');
6:   const context = canvas.getContext('2d');
7:
8:   context.fillStyle = 'green';
9:   context.fillRect(10, 10, 150, 100);
10: </script>
```

Resources: w3schools.com/tags/ref_canvas.asp



Lebih lanjut tentang Canvas API di tutorial dan penugasan



Terima Kasih

**Auzi Asfarian, M Asyhar Agmalaro,
Endang P Giri, Shervie N Neyman**

Departemen Ilmu Komputer, FMIPA
IPB University (Institut Pertanian Bogor), Bogor, Indonesia
asfarian@apps.ipb.ac.id | ux@apps.ipb.ac.id



**KOM1304 Grafika Komputer
dan Visualisasi**

Sistem Koordinat, Vertex dan Index, Garis

**Auzi Asfarian, M Asyhar Agmalaro,
Endang P Giri, Shervie N Neyman**

Departemen Ilmu Komputer, FMIPA
IPB University (Institut Pertanian Bogor), Bogor, Indonesia
asfarian@apps.ipb.ac.id | ux@apps.ipb.ac.id



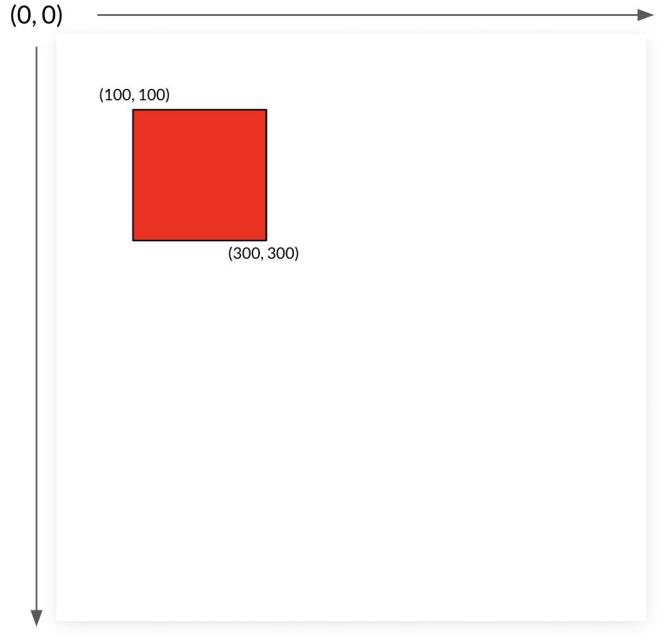
IPB University
— Bogor Indonesia —

Departemen
Ilmu Komputer

Bagian 1:

Sistem Koordinat

Sistem Koordinat



Koordinat pada kanvas.

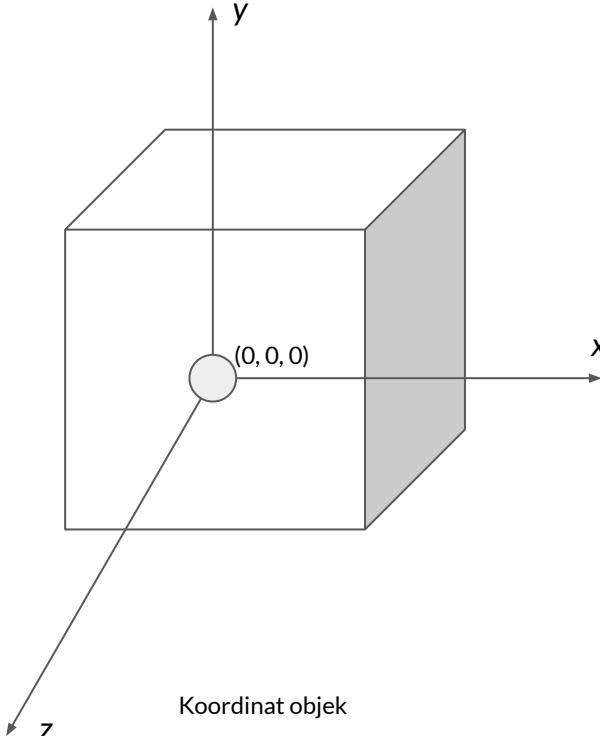
Suatu adegan (*scene*) grafika komputer melibatkan banyak objek dengan posisi relatifnya masing-masing.

Dibutuhkan suatu standar untuk mengatur peletakan tersebut.

Sistem koordinat Kartesius digunakan untuk menyatakan informasi posisi tersebut.

Koordinat Kartesius memiliki titik pusat (*center*) dan memiliki sumbu-x dan sumbu-y (untuk koordinat dua dimensi) dan sumbu-z (untuk koordinat tiga dimensi)

Sistem Koordinat

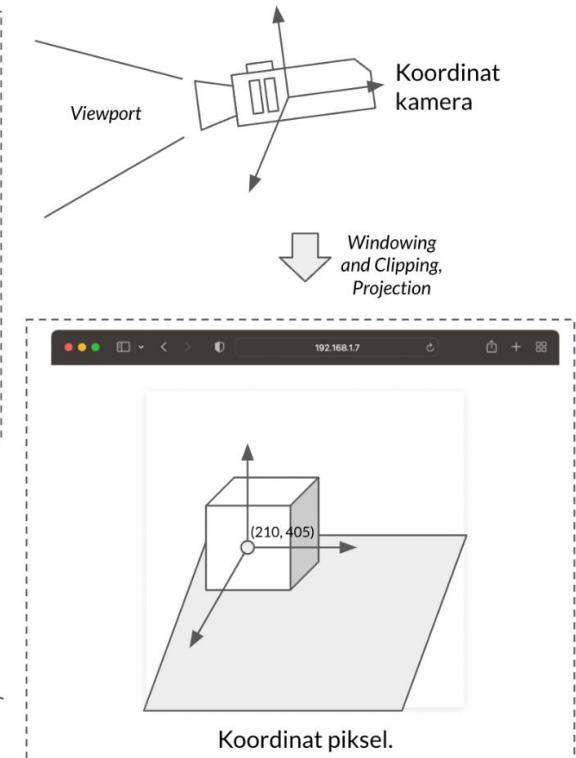
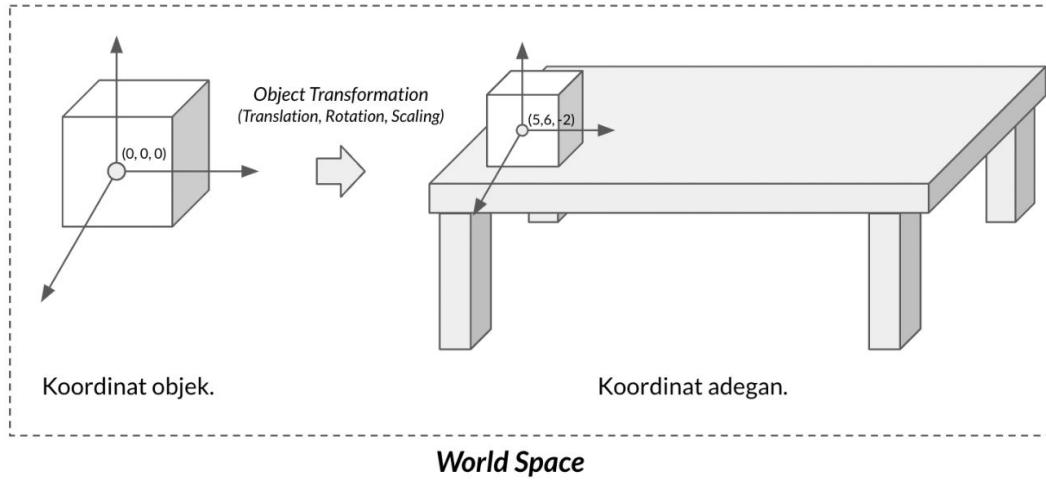


Sistem grafika komputer menggunakan beberapa jenis koordinat: objek, adegan, kamera, dan piksel.

Penggambaran objek geometri melibatkan proses konversi dan kalkulasi dari satu koordinat ke koordinat yang lain.

Salah satu tantangan adalah mayoritas perangkat display menampilkan gambar dalam bentuk dua dimensi.

Perubahan Koordinat pada Sistem Grafika Komputer



World Space: Representasi logika dari adegan grafis yang ada di dalam komputer.

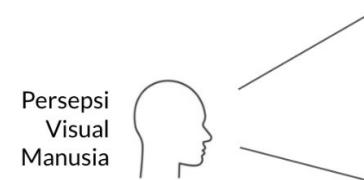


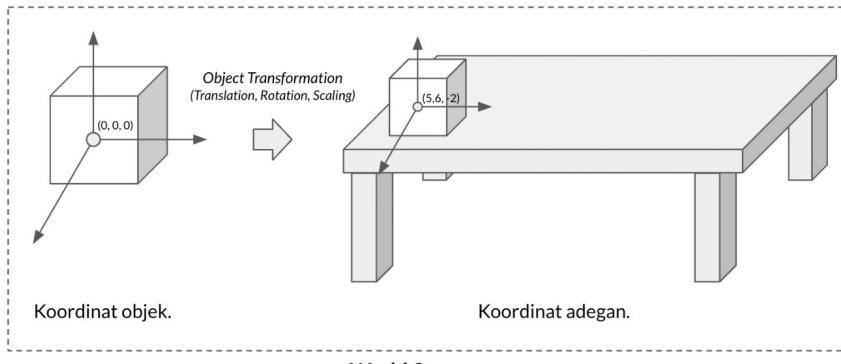
Image Space: Representasi visual dari adegan grafis yang siap ditampilkan ke dalam perangkat *display*.

World Space

Pada saat objek dibuat, koordinat yang digunakan adalah **koordinat objek**.

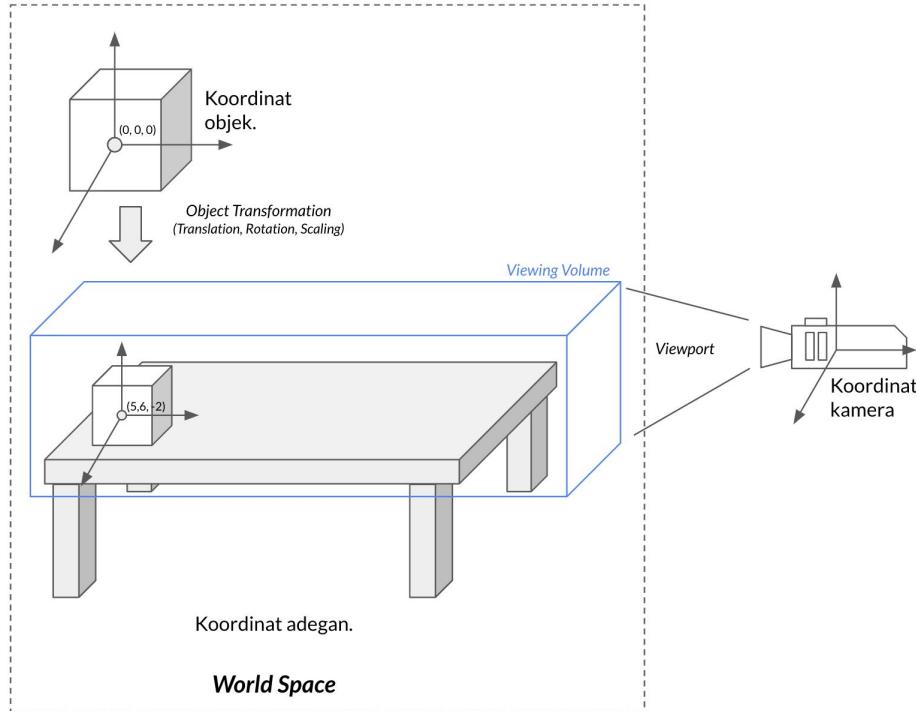
Setelah itu, objek akan ‘diletakkan’ bersama objek-objek lainnya dalam suatu adegan, yang memiliki **koordinat adegan**.

Proses peletakan melibatkan proses transformasi: translasi, rotasi, dan skalasi yang mengubah koordinat objek menjadi koordinat adegan.



Perhatikan perubahan titik pusat kubus dari koordinat objek ke koordinat adegan akibat proses transformasi yang dilakukan.

Dari *World Space* Menuju *Image Space*



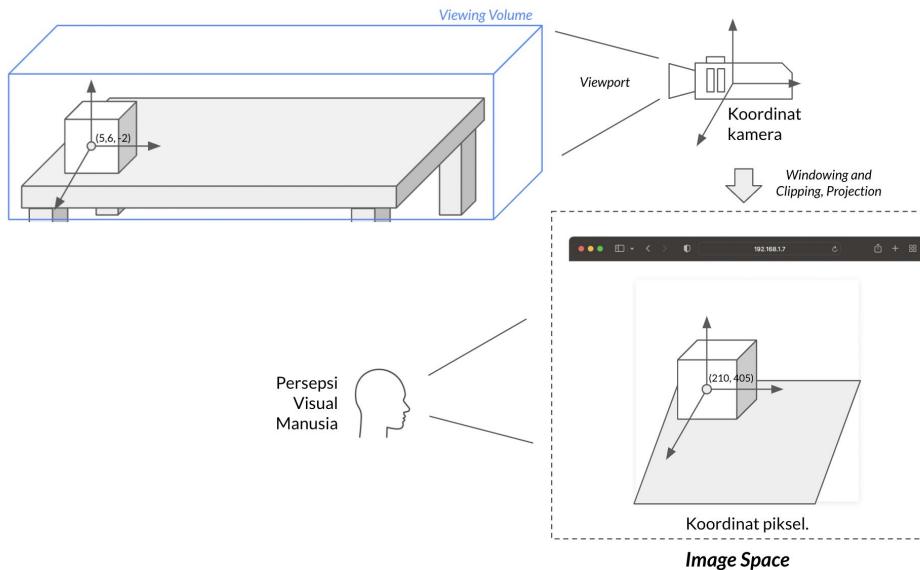
Seperti manusia yang memahami dunia melalui mata, sistem grafika komputer menggunakan kamera untuk 'merekam' *world space*.

Kamera dapat digerakkan dengan mengubah posisinya pada koordinat kamera.

World space yang akan diproses lebih lanjut dan ditampilkan di perangkat display adalah yang masuk pada *viewing volume* dari kamera.

Viewing volume diatur pada pengaturan *viewport* kamera.

Image Space

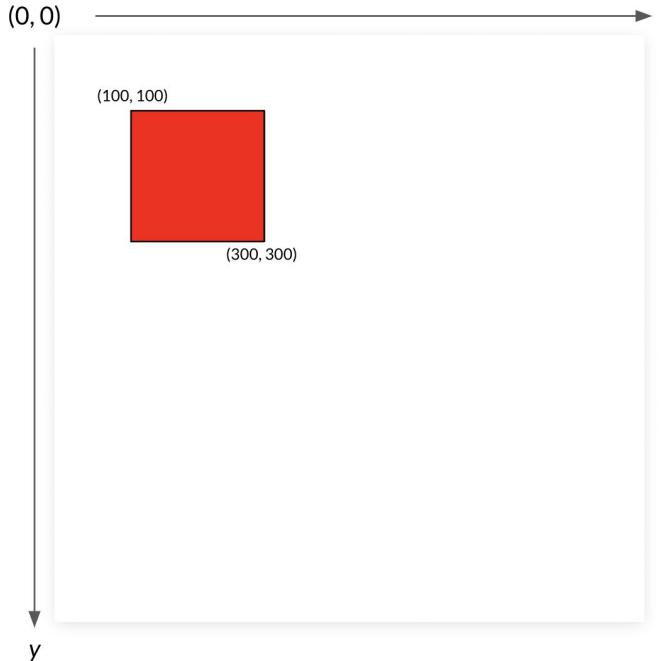


Viewing volume yang masih berupa tiga dimensi, harus diproyeksikan ke bidang dua dimensi untuk ditampilkan di perangkat *display*.

Proses ini melibatkan *windowing*, *clipping*, dan *projection* dan menghasilkan gambar raster dua dimensi yang siap ditampilkan di perangkat *display*.

Gambar tersebut ditampilkan dan dilihat oleh lewat mata manusia melalui sistem persepsi visual manusia.

Renungkan



Mengapa pada aktivitas sebelumnya kita hanya bekerja dengan satu koordinat Canvas?

Objek grafik dua dimensi lebih sederhana daripada tiga dimensi (misal: proyeksi tidak diperlukan).

Aktivitas tersebut masih sederhana sehingga koordinat objek, adegan, dan piksel disamakan untuk mengurangi kompleksitas.

Perbedaan koordinat-koordinat baru dirasakan saat kita bekerja dengan adegan tiga dimensi yang lebih kompleks.

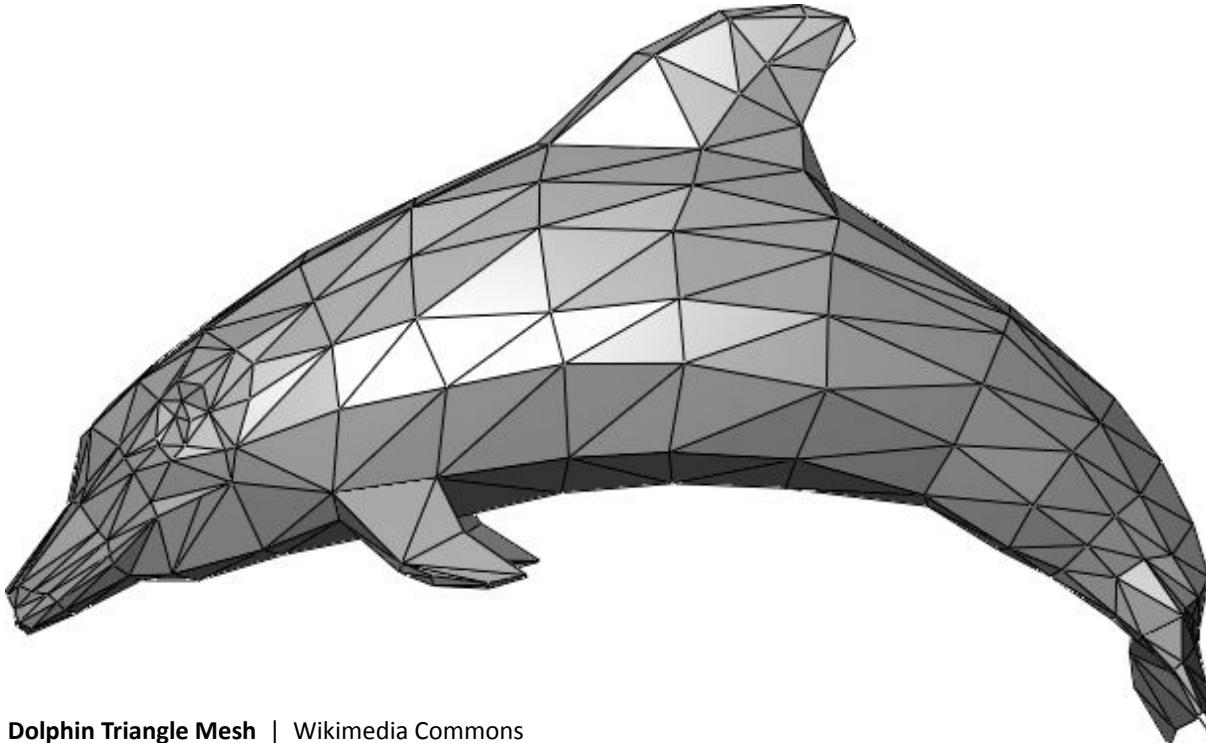


IPB University
— Bogor Indonesia —

Departemen
Ilmu Komputer

Bagian 2: *Vertex dan Index*

Setiap Objek Grafis Dibuat Berdasarkan Vertex



Dolphin Triangle Mesh | Wikimedia Commons

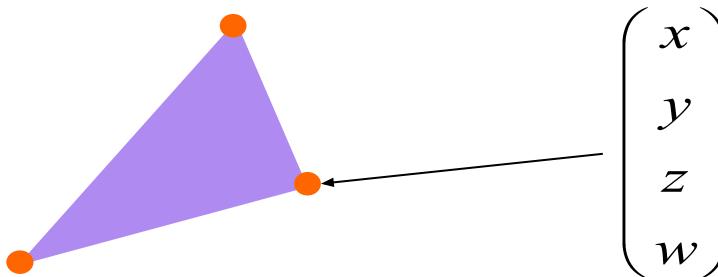
Arsitektur komputer tidak memungkinkan komputer mengolah informasi visual.

Oleh karena itu, informasi visual harus dideskripsikan menggunakan informasi numerik yang dipahami oleh komputer.

Vertex (jamak: vertices) adalah struktur data yang mendeskripsikan atribut pada objek grafis.

Semakin kompleks suatu objek, semakin banyak vertex yang menyusunnya.

Vertex



Verteks menyimpan data visual yang mendeskripsikan posisi (baik pada koordinat 2D atau 3D) dan data lain yang dibutuhkan untuk menggambar (*render*) objek tersebut.

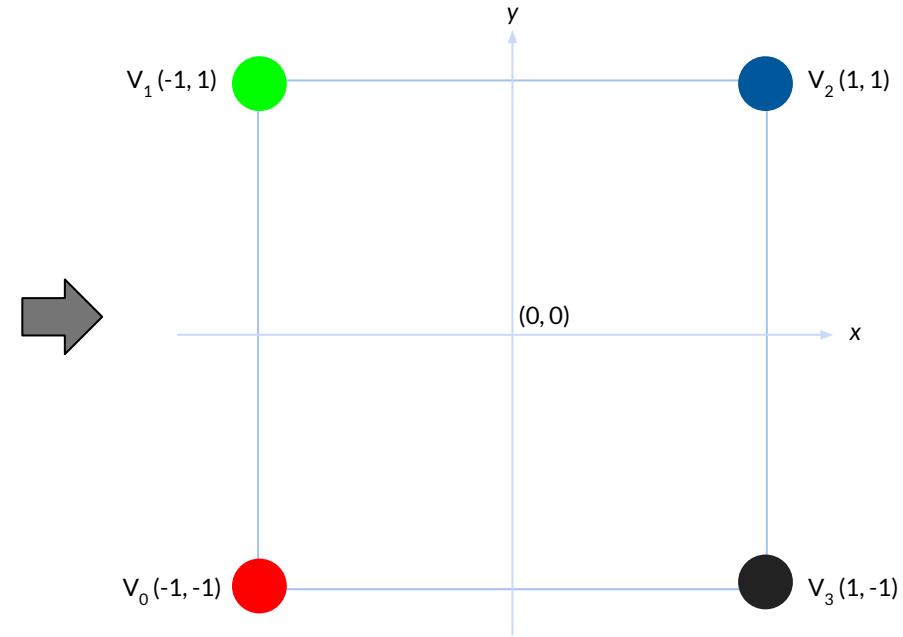
Data posisi: x , y , dan z .

Data lainnya: warna (*red*, *green*, *blue*), transparansi, pantulan, koordinat tekstur, dst.

Vertex dan Index

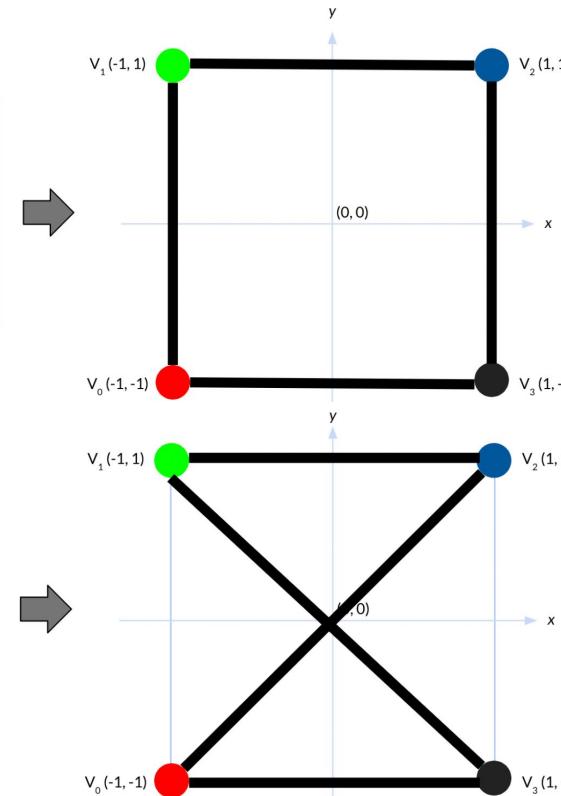
Sekumpulan verteks tidak cukup untuk mendeskripsikan suatu objek geometri.
Dibutuhkan satu informasi lain yang disebut sebagai *index*.

Verteks	x	y	Warna
0	-1	-1	#FF0000
1	-1	1	#00FF00
2	1	1	#0000FF
3	1	-1	#000000



Index

Verteks	x	y	Warna	Index
0	-1	-1	#FF0000	0
1	-1	1	#00FF00	1
2	1	1	#0000FF	2
3	1	-1	#000000	3

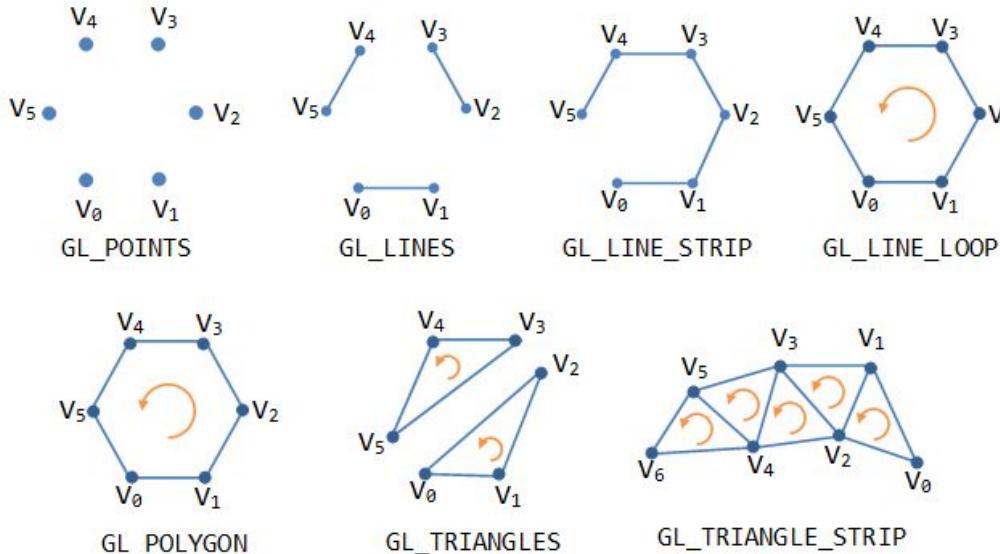


Indeks yang berbeda akan menghasilkan gambar yang berbeda (walaupun informasi vertex-nya sama)

Apabila tidak dispesifikasikan, API grafis secara *default* akan memberikan *index* 0 pada *vertex* yang pertama dibuat.

Oleh karena itu urutan pembuatan vertex akan sangat berpengaruh pada gambar yang akan dibuat.

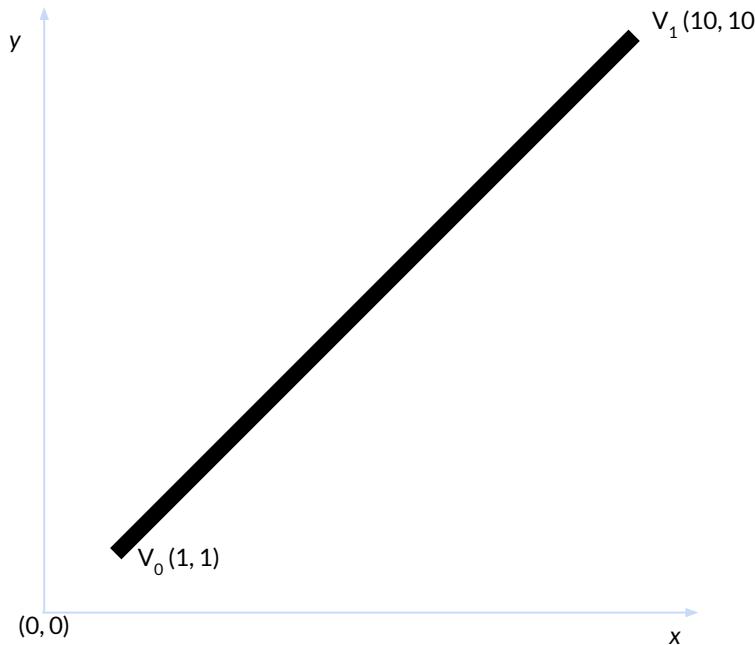
Mode Gambar (*Drawing Mode*)



Selain *vertex* dan *index*, satu informasi lain yang dibutuhkan adalah mode gambar.

Contoh sebelumnya digambar dengan menggunakan mode gambar **LINE_LOOP**.

Menggambar Garis

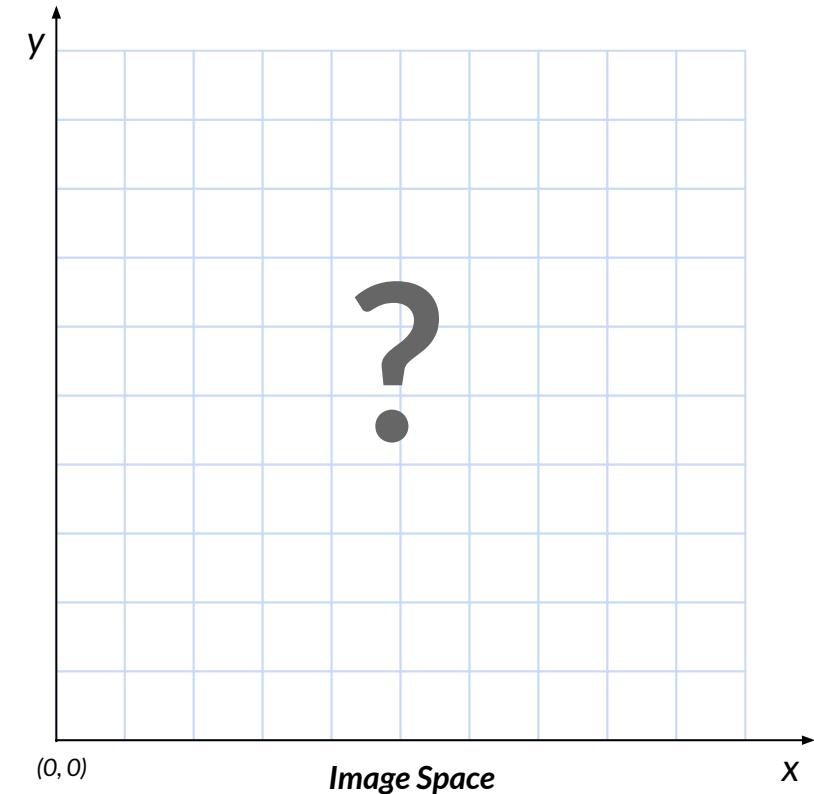
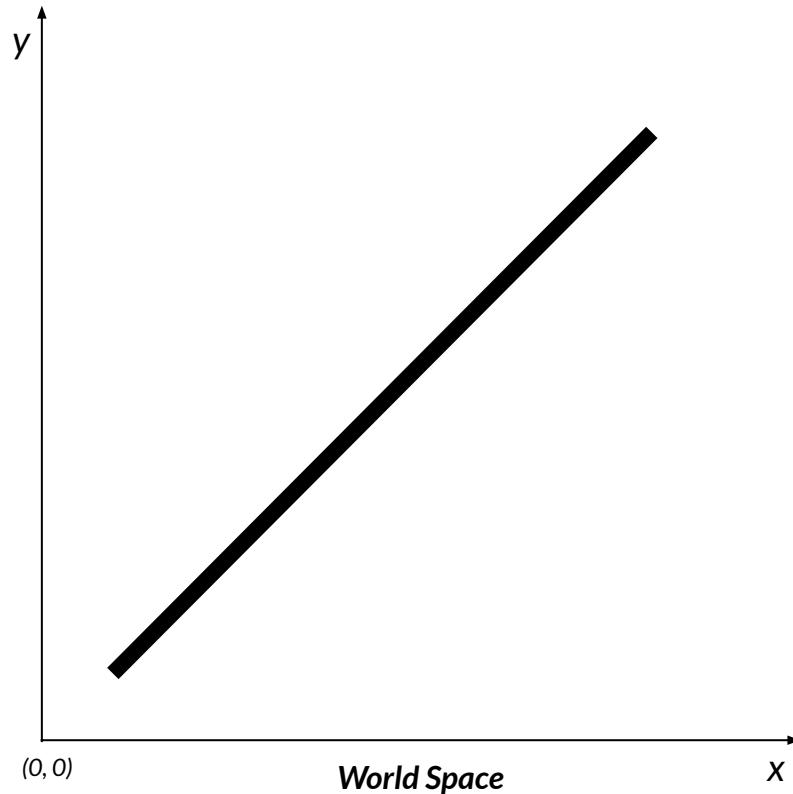


Garis sebagai objek geometri dasar (atau primitif) hanya membutuhkan dua buah vertex.

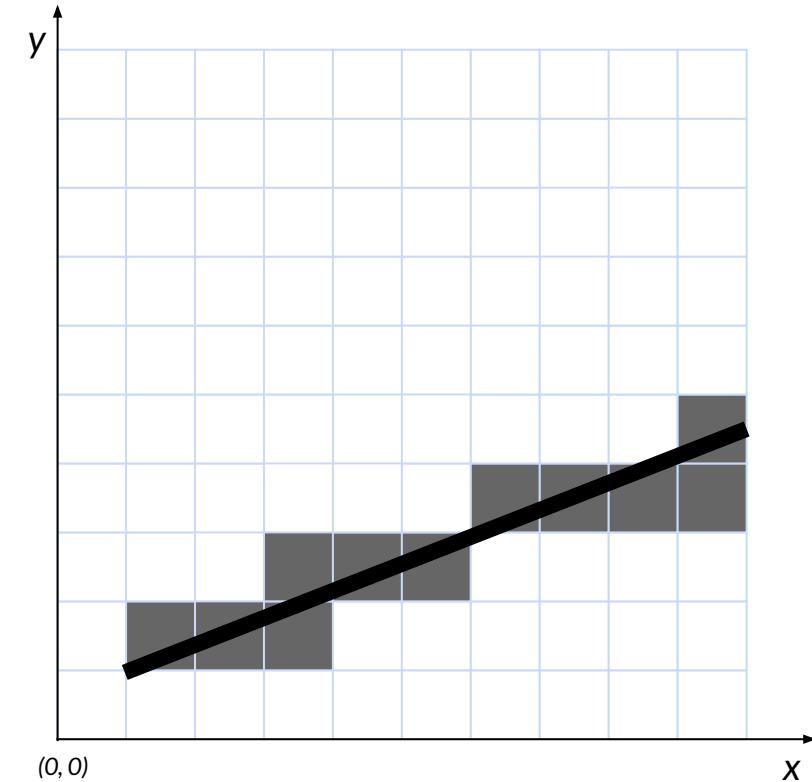
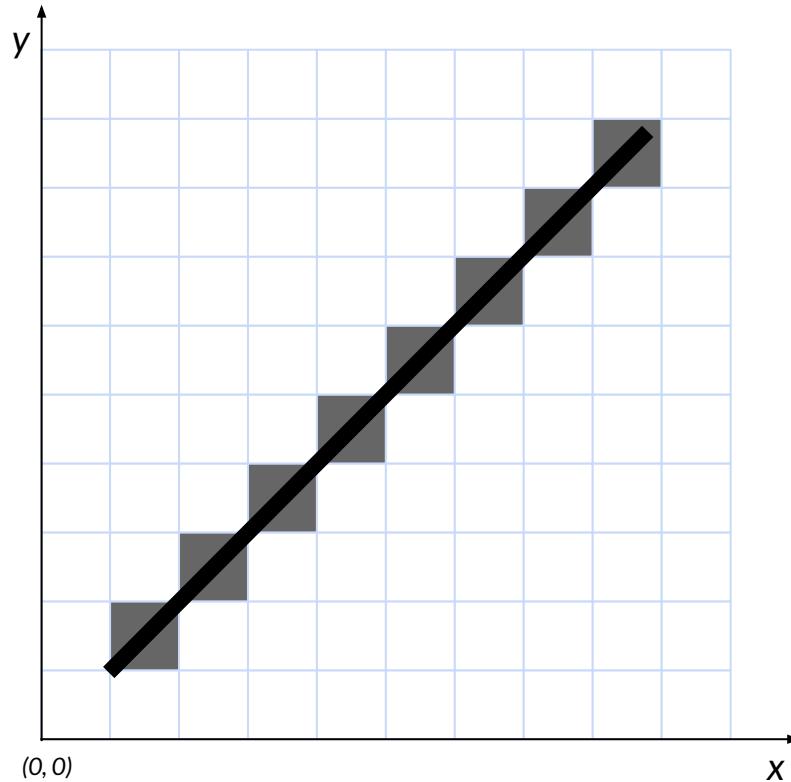
Secara logika dan implementasinya di kode program, menggambar sebuah garis sangat mudah dilakukan.

Akan tetapi, ingat bahwa satuan gambar di perangkat *display* adalah sebuah *pixel* yang berukuran kotak.

Menggambar Garis



Menggambar Garis





IPB University
— Bogor Indonesia —

Departemen
Ilmu Komputer

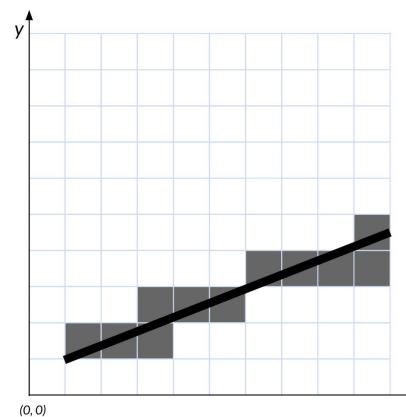
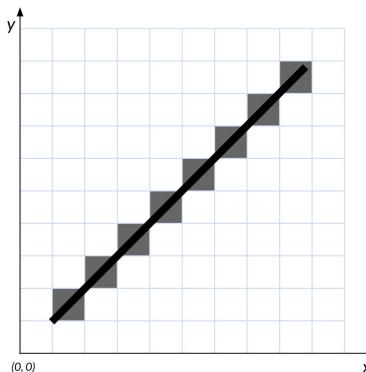
Bagian 3:

Algoritma Rasterisasi

Baris: Digital

Differential Analyzer

Algoritma untuk Menggambar Garis



Walaupun mudah untuk dilaksanakan oleh manusia, dibutuhkan suatu algoritma untuk menggambar garis secara konsisten.

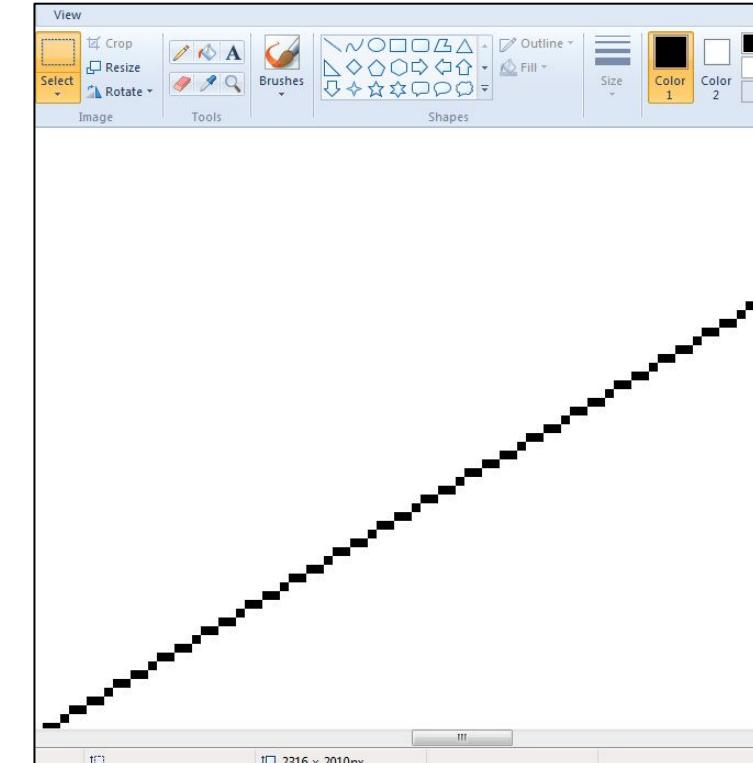
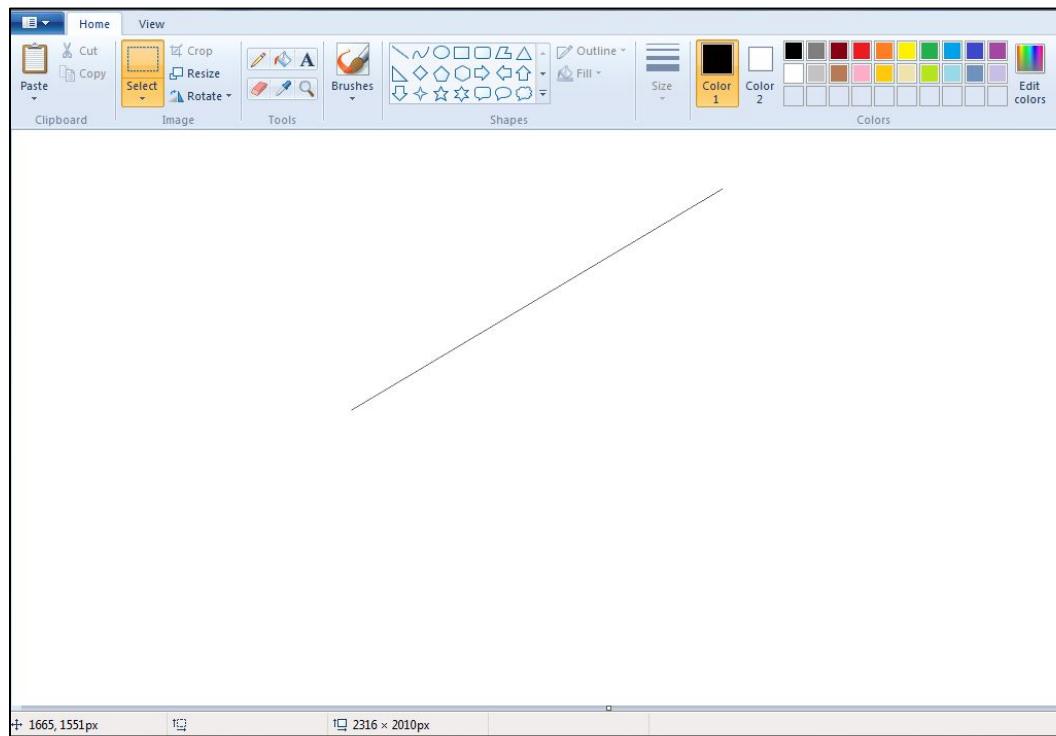
Oleh karena itu, garis perlu dilihat dalam bentuk persamaan garis.

Algoritma **line-scan** dapat digunakan untuk melakukan rasterisasi garis.

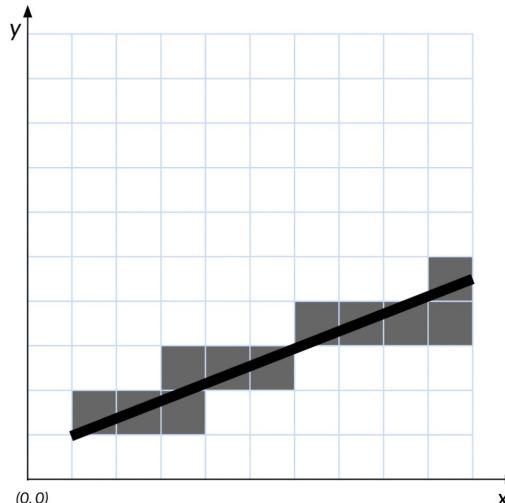
Line-scan akan menelusuri garis dari titik awal sampai akhir, dan mewarnai pixel yang mewakili garis.

Ada tiga algoritma line-scan dasar, yaitu *digital differential analyzer* (DDA), Bresenham, dan midpoint Bresenham.

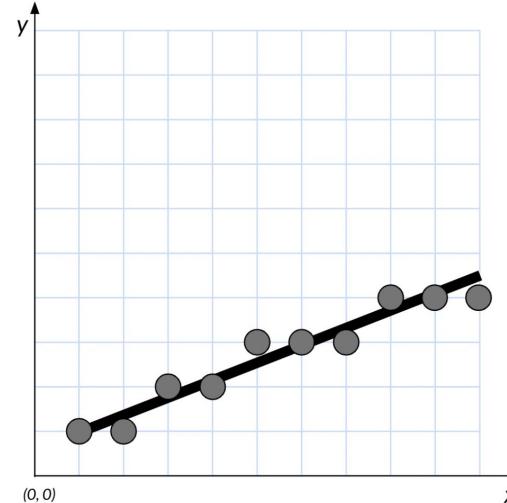
Algoritma untuk Menggambar Garis



Koordinat Piksel vs Koordinat Kartesius



Koordinat Piksel



Koordinat Kartesius

Untuk memperlihatkan cara kerja algoritma dengan lebih jelas, gambar garis akan digambar pada koordinat kartesius.

Titik pada koordinat kartesius melambangkan titik yang mewakili garis yang akan digambar.

Pada setiap kolom, ada satu titik yang digambar.

Konsep Dasar Menggambar Garis

Garis

- Dinyatakan dengan empat titik dengan masing-masing : $x_0, y_0, x_{end}, y_{end}$
- Koordinat / titik awal (x_0, y_0)
- Koordinat / titik akhir (x_{end}, y_{end})
- Dari keempat titik tersebut dapat dihasilkan sebuah **gradien/slope/kemiringan garis** yang akan digambarkan (m), dengan mengambil nilai selisih dari nilai titik masing-masing di ordinat dan absisnya.

$$m = \frac{y_{end} - y_0}{x_{end} - x_0}$$

Konsep Dasar Menggambar Garis

Garis

- Menggunakan konsep dasar dari kalkulus, maka dengan menggunakan gradien(m) tersebut maka kita dapat menghasilkan sebuah persamaan garis lurus yang dinyatakan dengan:

$$y = mx + b$$

dengan m merupakan kemiringan garis dan b suatu nilai konstanta

- Dengan melihat karakteristik dari nilai m , maka persamaan garis yang dihasilkan dapat dikelompokkan ke dalam 3 bentuk:
 - Garis **cenderung tegak**, jika nilai $m > 1$,
 - Garis **cenderung landai**, jika nilai $m < 1$, dan
 - Garis **miring 45 derajat**, jika nilai $m = 1$

Konsep Dasar Menggambar Garis

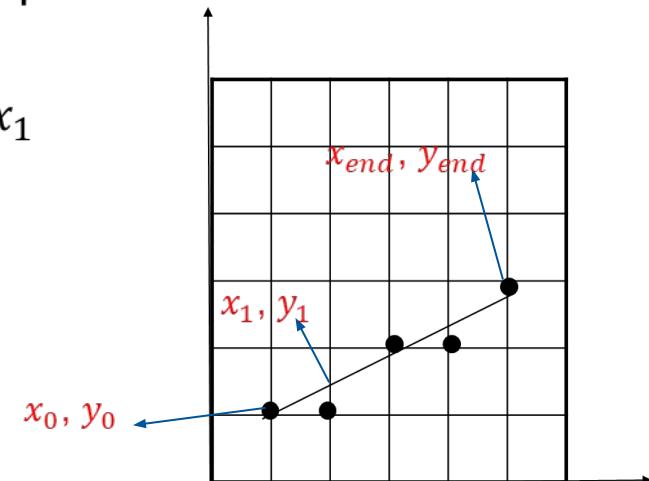
Garis

- dari persamaan $y = mx + b$, dan karena kita tahu sepanjang garis **nilai b akan selalu konstan**, maka jika kita masukkan titik-titik yang melewati garis misalnya (x_0, y_0) , dan (x_1, y_1) kita akan dapatkan:

$$b = y_0 - mx_0 = y_1 - mx_1$$

atau:

$$y_1 = m(x_1 - x_0) + y_0$$



Konsep Dasar Menggambar Garis

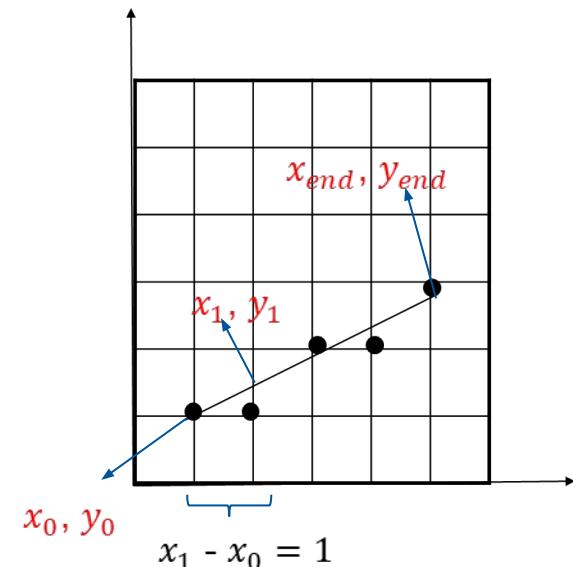
Garis

- dari persamaan $y_1 = m(x_1 - x_0) + y_0$ kita mengetahui bahwa nilai $\Delta x = x_1 - x_0 = 1$
maka secara sederhana kita dapat menghitung nilai

$$y_1 = m + y_0$$

atau secara **increment** jika nilai $i = 0$, maka untuk i selanjutnya akan didapatkan nilai y secara rekursif:

$$y_{i+1} = m + y_i$$



Konsep Dasar Menggambar Garis

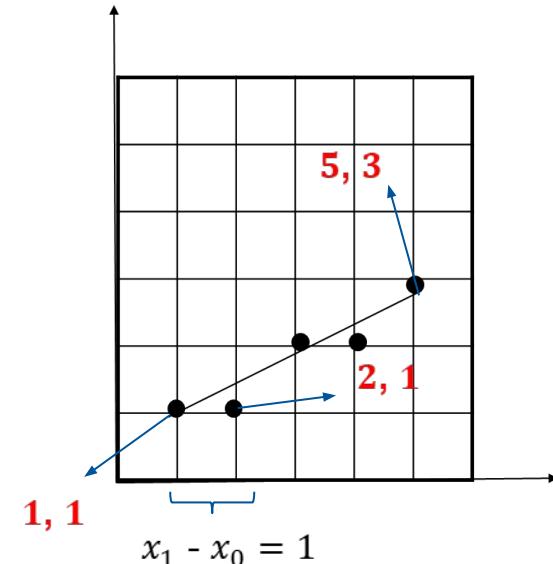
Garis (Contoh)

- Koordinat / titik awal ($x_0 = 1, y_0 = 1$)
- Koordinat / titik akhir ($x_{end} = 5, y_{end} = 3$)
- $m = \frac{y_{end}-y_0}{x_{end}-x_0} = \frac{3-1}{5-1} = 0.5 < 1 \rightarrow \text{Cenderung Landai}$
untuk y_1 didapatkan secara rekursif:

$$y_1 = m + y_0 = 0.5 + 1 = 1.5$$

sehingga didapatkan titik $x_1 = 2, y_1 = 1.5$

namun menjadi catatan untuk representasi koordinat layar maka nilai y_1 dikonversikan ke pembulatan integer menjadi $x_1 = 2, y_1 = 1$ dan seterusnya sampai didapatkan $x_{end} = 5$ dan $y_{end} = 3$



Algoritma Digital Differential Analyzer (DDA) untuk Menggambar Garis

- DDA adalah algoritme pembentuk garis yang didasarkan dari **kONSEP PENGAMBARAN GARIS**, dimana garis dibuat menggunakan titik awal (x_0, y_0) dan titik akhir (x_{end}, y_{end}).
- Setiap koordinat titik (x_i, y_i) yang membentuk garis diperoleh dari perhitungan, yang kemudian hasil perhitungan tersebut dikonversikan menjadi nilai integer(rounding)
- Kelemahan : Komputasi tinggi, tidak efisien

Costly floating point computations !!
Multiplications,
additions, roundings

Algoritma Digital Differential Analyzer (DDA) untuk Menggambar Garis

- **Algoritme DDA** dapat digunakan untuk menghitung garis dengan kemiringan, baik dengan garis yang memiliki kemiringan landai($m < 1$) atau yang cenderung tegak($m > 1$)
- Untuk garis dengan **kemiringan landai**, maka gunakan **sumbu-x sebagai iterator** (nilai x bertambah/increment 1), sedangkan **untuk nilai y dievaluasi dengan: $y_{i+1} = m + y_i$** , kemudian nilai **y tersebut dikonversi ke nilai integer(rounding)**
- Untuk garis dengan **kemiringan cenderung tegak**, maka gunakan **sumbu-y sebagai iterator** (nilai y bertambah/increment 1) sedangkan untuk nilai **x dievaluasi dengan: $x_{i+1} = \frac{1}{m} + x_i$** , kemudian nilai **x tersebut dikonversi ke nilai integer(rounding)**

Algoritma Digital Differential Analyzer (DDA) untuk Menggambar Garis

- Berikut adalah langkah-langkah Algoritme DDA:
 1. Tentukan 2 titik yang dihubungkan untuk membentuk garis sebagai menggunakan titik awal (x_0, y_0) dan titik akhir (x_{end}, y_{end})
 2. Hitung $D_x = x_{end} - x_0$ dan $D_y = y_{end} - y_0$ serta $m = \frac{D_y}{D_x}$
 3. Tentukan **iterator** dan Jumlah **step**, dan nilai incerement dengan kententuan:
bila $|D_y| < |D_x|$ **Maka iterator adalah sumbu-x dengan jumlah step:** $|D_x|$ **serta** $y_incre = m$, selainnya maka iterator adalah sumbu-y dengan jumlah step $|D_y|$ **dan** $x_incre = \frac{1}{m}$
 4. Next Koordinat, jika iterator sumbu-x adalah $y = y + y_incre$; dan $round(y)$ selainnya $x = x + x_incre$; dan $round(x)$
 5. Ulangi langkah 4 untuk menentukan posisi pixel sampai $x = x_{end}$ dan $y = y_{end}$

Algoritma Digital Differential Analyzer (DDA) untuk Menggambar Garis

Algoritma DDA:

```
compute (m);
if m < 1:
    float y = y0;          // initial value
    for(int x = x0;x <= x1; x++, y += m)
        setPixel(x, round(y));
else if m > 1:
    float x = x0;          // initial value
    for(int y = y0;y <= y1; y++, x += 1/m)
        setPixel(round(x), y);
```

Catatan:

setPixel(x, y) menggambar piksel pada kolom x dan baris y.

Contoh

- Contoh DDA ($m < 1$):

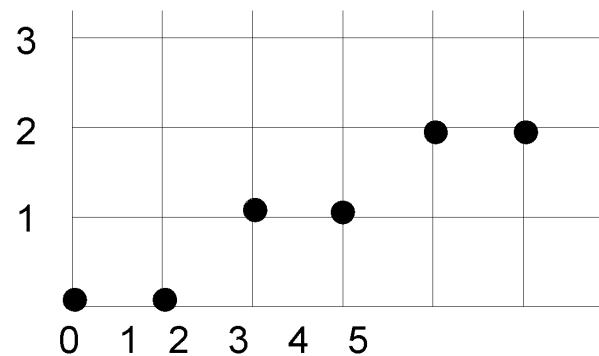
$$P_0 = (0,0) \rightarrow P_{end} = (5,2) \quad D_x = 5 \text{ dan } D_y = 2 \text{ serta } m = \frac{2}{5} = 0.4$$

$|D_y| < |D_x| \rightarrow$ iterator sumbu-x dengan jumlah **step = 5** dan **y_incre = 0.4**

initial $x, y = (0,0)$, next x, y :

round(y)

Line: $P_0(0, 0) \text{-- } P_1(5, 2)$



step	x	y=y+y_inc	int(y+0.5)
1	1	0+0.4	0
2	2	0.4+0.4	1
3	3	0.8+0.4	1
4	4	1.2+0.4	2
5	5	1.6+0.4	2

Latihan

- **Latihan/Tugas 1**

1. *Diketahui 2 buah titik A(10,10) dan titik B(17,16), bila titik A sebagai titik awal dan titik B sebagai titik akhir maka buatlah/gambarkan garis yang menghubungkan titik tersebut dengan menggunakan algoritma DDA.*
2. *Diketahui 2 buah titik A(10,10) dan titik B(16,17), bila titik A sebagai titik awal dan titik B sebagai titik akhir maka buatlah/gambarkan garis yang menghubungkan titik tersebut dengan menggunakan algoritma DDA.*



Terima Kasih

**Auzi Asfarian, M Asyhar Agmalaro,
Endang P Giri, Shervie N Neyman**

Departemen Ilmu Komputer, FMIPA
IPB University (Institut Pertanian Bogor), Bogor, Indonesia
asfarian@apps.ipb.ac.id | ux@apps.ipb.ac.id



**KOM1304 Grafika Komputer
dan Visualisasi**

Algoritme Midpoint untuk Garis dan Lingkaran/Elips

**Auzi Asfarian, M Asyhar Agmalaro,
Endang P Giri, Shelvie N Neyman**

Departemen Ilmu Komputer, FMIPA
IPB University (Institut Pertanian Bogor), Bogor, Indonesia
asfarian@apps.ipb.ac.id | ux@apps.ipb.ac.id



IPB University
— Bogor Indonesia —

Departemen
Ilmu Komputer

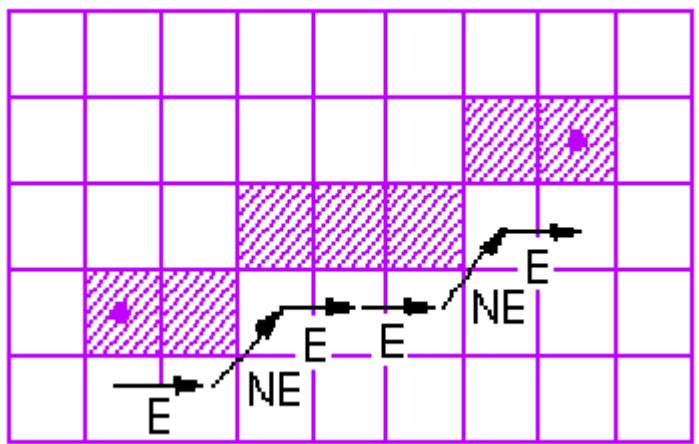
Bagian 1: Midpoint Line

Algoritme Midpoint

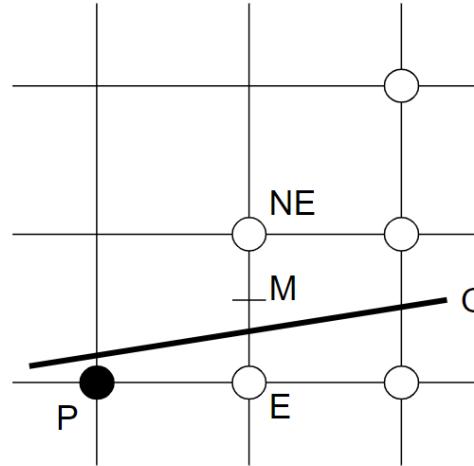
Algoritme midpoint merupakan perbaikan dari algoritme bresenham yang umum dikarenakan meminimalkan penghitungan yang melibatkan float dan hanya menggunakan penghitungan integer.

Algoritme midpoint memposisikan penggambaran garis hanya sebagai rangkaian/skuens dari keputusan (decisions).

Untuk setiap pixel yang digambarkan, akan memberikan informasi pixel berikutnya dengan keputusan berlabel E (east) atau NE(North East), seperti ilustrasi gambar



Algoritme Midpoint



Cara algoritme bresenham dalam menentukan kandidat titik pixel (dengan label E dan NE) berdasarkan jarak terdekat dari garis/line ke **titik kandidat E atau NE** :

```
If distance(E, Q) < distance(NE, Q)  
then
```

```
    select E = (xp+1, yp)  
else  
    select NE = (xp+1, yp+1)
```

Berbeda dengan Bresenham, algoritme midpoint memanfaatkan evaluasi titik tengah (**M**) yang berada di atas garis atau berada di bawah garis sebagai parameter keputusan untuk memilih titik **E** atau **NE**.

Algoritme Midpoint

Dalam bentuk implisit, persamaan garis dapat diformulasikan ke dalam bentuk:

$$F(x, y) = ax + by + c = 0$$

Dengan menggunakan persamaan sederhana

$$y = mx + b; \text{ dimana } m = \frac{D_y}{D_x}; \text{ menjadi:}$$

$$y = \frac{D_y}{D_x}x + b$$

kemudian menghilangkan pecahan menjadi:

$$D_x y = D_y x + D_x b \rightarrow D_y x - D_x y + \Delta x b = 0$$

menjadi:

$$F(x, y) = D_y x - D_x y + \Delta x b = 0; \text{ dimana } a = D_y, b = -D_x, \text{ dan } c = \Delta x b$$

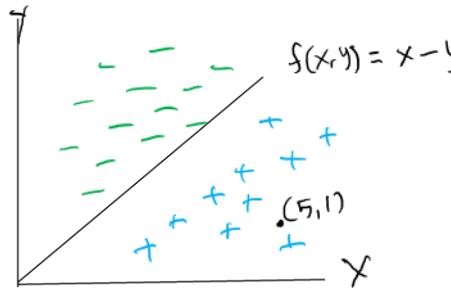
Algoritme Midpoint

Dengan membuat Asumsi bahwa kemiringan dengan $b < 0$, maka bisa diimplikasikan:

Nilai $F(x, y) > 0$ jika titik-titik yang dievaluasi terhadap fungsi **terletak di bawah garis**, dan sebaliknya $F(x, y) < 0$ jika titik-titik yang dievaluasi ke fungsi **terletak di atas garis**.

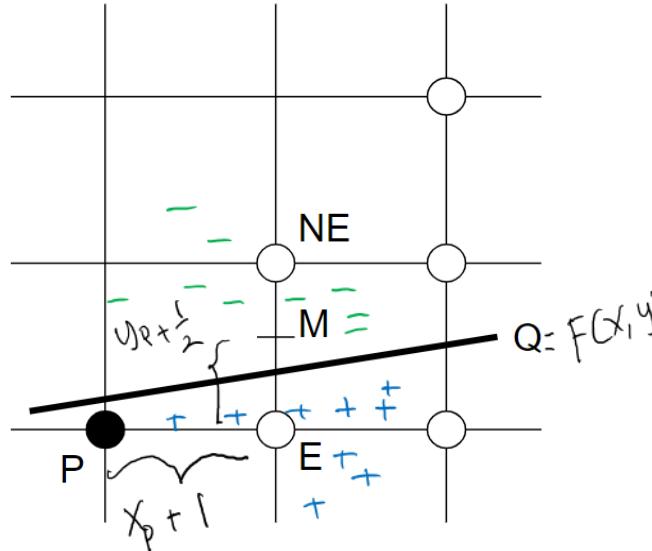
Sebagai contoh $F(x, y) = x - y = 0$ didapatkan dari persamaan sederhana $y = x$.

Jika kita lihat pada grafik:



Kemudian diambil titik di bawah grafik misalkan $(5, 1)$ kemudian **dievaluasi** ke fungsi $F(x, y) = x - y = 5 - 1 = 4$ (**positif**) dan jika diambil titik lain di bawah garis maka akan selalu menghasilkan nilai positif, sebaliknya jika diambil titik di atas garis dan dievaluasi ke fungsi akan menghasilkan nilai **negatif**

Algoritme Midpoint



Pada Algoritme midpoint, maka kriteria titik tengah **M** digunakan sepanjang garis untuk melihat apakah terletak di atas garis atau dibawah garis yang memberikan 2 keputusan negative atau positif sebagai parameter keputusan.

Untuk mengaplikasikan kriteria midpoint, secara sederhana dilakukan dengan menghitung dan mengevaluasi titik **M** ke dalam Fungsi $F(x, y)$ sebagai parameter keputusan memilih kandidat pixel E atau NE.

$$F(M) = F\left(x_p + 1, y_p + \frac{1}{2}\right)$$

Algoritme Midpoint

Untuk menentukan titik pixel mana yang akan dipilih maka dibutuhkan sebuah variabel keputusan yang ditentukan dengan mengevaluasi nilai M pada fungsi F $D = F(M)$ kemudian variable D digunakan untuk menentukan titik terpilih dengan melihat tanda yang dihasilkan negatif/positif.

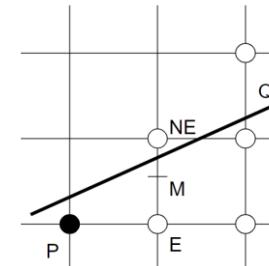
If $D > 0$ then M ada berada di bawah garis, maka titik yang **dipilih adalah NE**, selainnya makan titik ada di atas garis dan **pilih E**

Titik tengah $M = (xp + 1, yp + \frac{1}{2})$ selanjutnya M dievaluasi ke fungsi F sebagai parameter $D = F(M)$

Dimisalkan $F(x, y) = 2ax + 2bx + c$ (nilai 2 ditambahkan untuk mengurangi kompleksitas hitungan pecahan)

$$D = F(xp + 1, yp + \frac{1}{2})$$

$$\begin{aligned} D &= 2a(xp + 1) + 2b(yp + \frac{1}{2}) + 2c \\ &= 2axp + 2byp + 2c + 2a + b \end{aligned}$$



Algoritme Midpoint

Karena kita ketahui bahwa titik penggambaran dimulai pada (x_0, y_0) yang terletak tepat pada garis maka jika dievaluasi

$$F(x_0, y_0) = 2ax_0 + 2bx_0 + 2c = 0$$

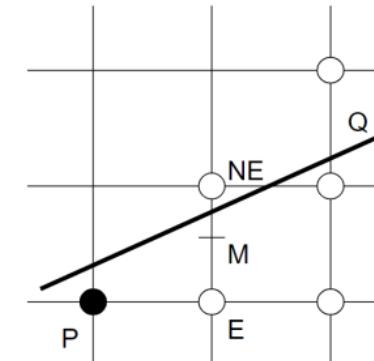
Parameter inisial D dapat dikalkulasi dengan:

$$\begin{aligned} D_{init} &= F(x_0 + 1, y_0 + \frac{1}{2}) \\ &= (2ax_0 + 2by_0 + 2c) + (2a + b) \\ &\quad \underbrace{}_0 \\ &= 0 + 2a + b \end{aligned}$$

Maka $D_{init} = 2a + b$

Advantages:

Dapat dilakukan dengan operasi shift (menambahkan integer dan mengalikan dengan 2), serta incremental.



Algoritme Midpoint

Bagaimana menghitung D selanjutnya secara incremental? Jika diketahui nilai D awal atau D_{init} ?

- Jika yang dituju adalah **pixel berlabel E**, maka D selanjutnya:

$$\begin{aligned} D_{new} &= F(x_p + 2, y_p + \frac{1}{2}) \\ &= 2a(x_p + 2) + 2b(y_p + \frac{1}{2}) + 2c \\ &= \dots \dots \dots \text{(untuk penjabaran lihat video di newlms)} \\ &= D + 2a \end{aligned}$$

- Jika yang dituju adalah **pixel berlabel NE**, maka D selanjutnya:

$$\begin{aligned} D_{new} &= F(x_p + 2, y_p + 1 + \frac{1}{2}) \\ &= 2a(x_p + 2) + 2b(y_p + 1 + \frac{1}{2}) + 2c \\ &= \dots \dots \dots \text{(untuk penjabaran lihat video di newlms)} \\ &= D + 2(a + b) \end{aligned}$$

Algoritme Midpoint

Langkah-langkah Algoritme Midpoint:

1. Hitung konstanta $a = y_1 - y_0$
 $b = x_0 - x_1$
2. Hitung inisial parameter D : $D_{init} = 2a + b$, lihat tandanya negatif atau positif
3. Untuk iterasi sebanyak Δx :

Jika $D < 0$

maka pilih **label E**, dan update titik $x_{i+1} = x_i + 1$ sedangkan y tetap y_i
Selanjutnya update $D_{new} = D_{old} + 2a$

Selainnya jika $D \geq 0$

maka pilih **label NE**, dan update titik $x_{i+1} = x_i + 1$ dan $y_{i+1} = y_i + 1$
Selanjutnya update $D_{new} = D_{old} + 2(a + b)$

Algoritme Midpoint

Contoh Algoritme Midpoint:

Gambarkan sebuah garis dengan algoritme midpoint dari titik awal (0,0) dan titik akhir (5,2)

Hitung dulu nilai berikut:

$$\begin{aligned}D_y &= y_n - y_0 = 2 - 0 = 2 \rightarrow a = 2 \\D_x &= x_n - x_0 = 5 - 0 = 5 \rightarrow b = -5 \\D_{init} &= 2a + b = 2(2) - 5 = -1 \\inc_E &= 2 * D_y = 4 \\inc_NE &= 2 * (D_y + D_x) = -6\end{aligned}$$

Kemudian lakukan pengecekan berikut:

```
if (D>0)
{
    x++; y++; //titik selanjutnya pilih titik atas
    D = D + inc_NE;
}
else {
    x++; //titik selanjutnya pilih titik bawah
    D = D + inc_E;
```

i	x_i	y_i	D	Keterangan
init	0	0	-1	Inisialisasi, $D<0$, E
1	1	0	3	y tetap, $D>0$, NE
2	2	1	-3	x,y incre, $D<0$, E
3	3	1	1	y tetap, $D>0$, NE
4	4	2	-5	x,y incre, $D<0$, E
5	5	2	-	Berhenti.



IPB University
— Bogor Indonesia —

Departemen
Ilmu Komputer

Bagian 2: *Midpoint Circle & Elips*

Algoritme Midpoint untuk Lingkaran

Dikarenakan Algoritme midpoint merupakan generalisasi dari algoritme Bresenham, maka selain untuk penggambaran garis, Midpoint juga dapat digunakan untuk objek-objek lain seperti lingkaran dan elips,

Caranya sama jika pada garis maka algoritme mengevaluasi titik tengah terhadap fungsi line, maka untuk objek lingkaran dan elips titik tengah dievaluasi terhadap fungsi circle/elips

Misalkan persamaan fungsi lingkaran dengan r sebagai jari-jari lingkaran sbb :

$$x^2 + y^2 = r^2$$

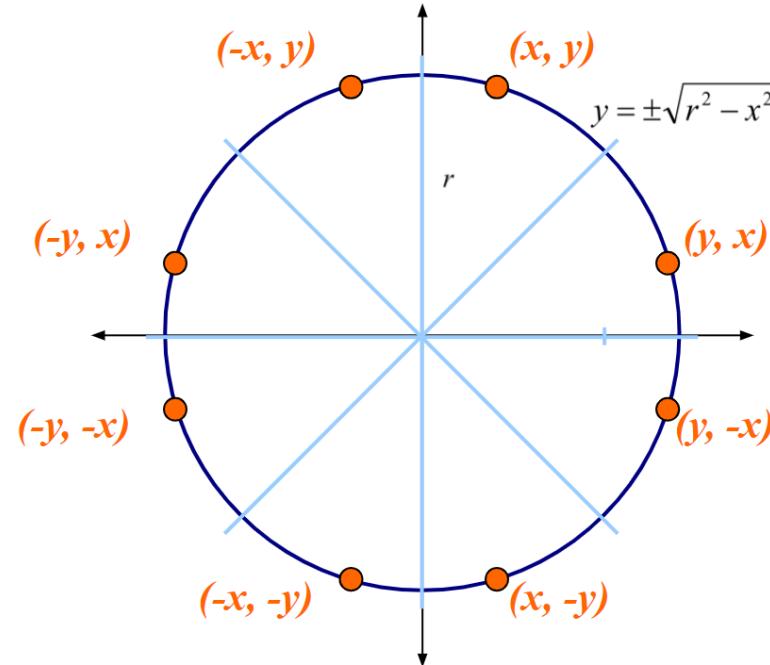
Jadim dapat dibuat algoritme penggambaran sederhana sebuah lingkaran dengan mencari solusi persamaan y pada interval $-x$:

$$y = \pm\sqrt{r^2 - x^2}$$

Untuk algoritme penggambaran lingkaran, pusat lingkaran harus berada pada titik origin di $(0,0)$ dengan konsep *eight-way symmetry*

Eight-way symmetry

Pada konsep eight-way symmetry menyatakan bahwa penggambaran lingkaran hanya cukup dilakukan pada salah satu kuadran pada sistem koordinat atau 1/8 saja bagian lingkaran, sedangkan untuk titik-titik lain dilakukan refleksi dari bagian yang di gambarkan.



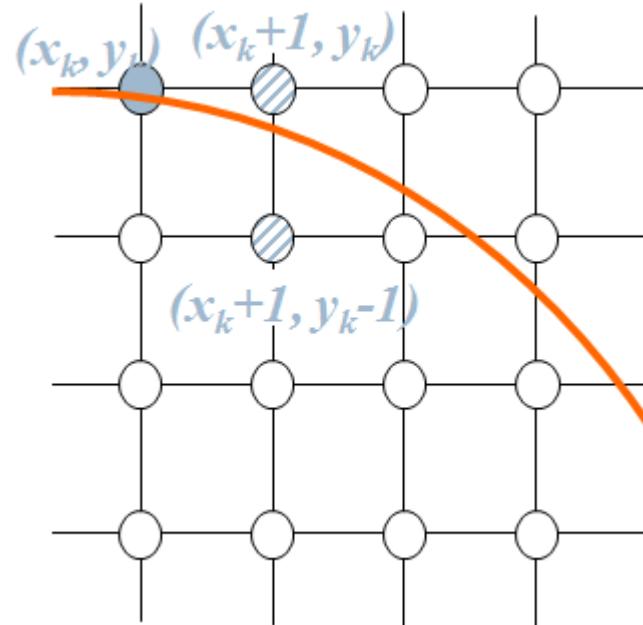
Midpoint Circle

Asumsikan bahwa titik(vertex) untuk penggambaran garis dimulai dari titik (x_k, y_k)

Titik selanjutnya adalah bergantung kepada dua pilihan yaitu $(x_k + 1, y_k)$ atau $(x_k + 1, y_k - 1)$

Jika dilihat pada gambar, titik selanjutnya yang akan dipilih adalah yang dekat dengan lingkaran. Bagaimana Caranya?

Sama dengan midpoint garis yang mengevaluasi titik tengah (M) berada di atas/di bawah fungsi garis, maka lingkaran akan mengevaluasi titik M berada di dalam/ di luar fungsi lingkaran.



Midpoint Circle

Diketahui persamaan Lingkaran adalah sebagai berikut:

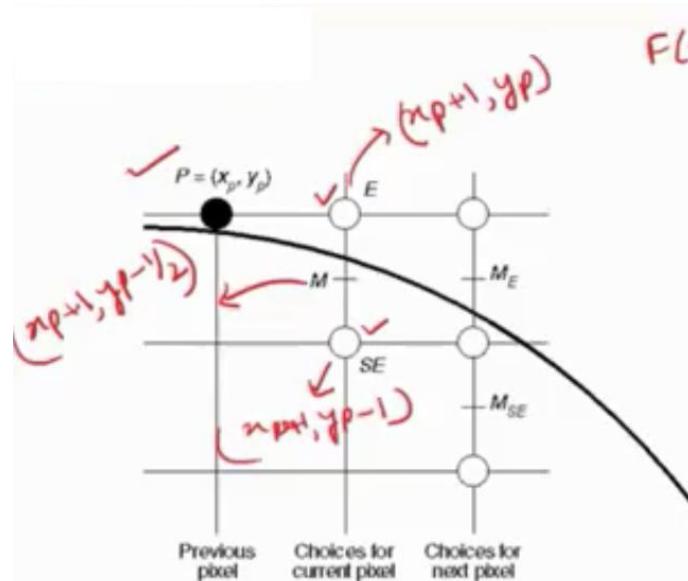
$$f_{circ}(x, y) = x^2 + y^2 - r^2$$

Dengan mengevaluasi titik tengah (M) terhadap fungsi lingkaran maka akan diketahui kandidat pixel yang akan dipilih, serta untuk mendefinisikan parameter keputusan yang menjadi discriminator.

$$f_{circ}(x, y) \begin{cases} < 0, & \text{jika titik } (x, y) \text{ dalam batas lingkaran} \\ = 0, & \text{jika titik } (x, y) \text{ tepat pada lingkaran} \\ > 0, & \text{jika titik } (x, y) \text{ di luar batas lingkaran} \end{cases}$$

Berbeda dengan garis, label pixel terpilih pada lingkaran dinamai dengan E (east), dan SE (South East)

Midpoint Circle



$$F(x, y) = x^2 + y^2 - R^2 = 0$$

$\boxed{E} / \boxed{SE} \rightarrow M$ (mid point)

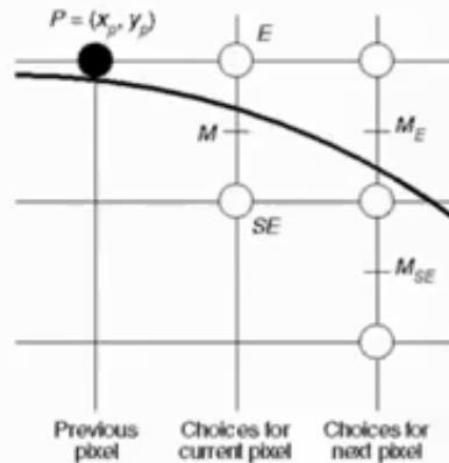
$F(x, y) > 0 \rightarrow$ point outside the circle

$F(x, y) < 0 \rightarrow$ point inside the circle.

$$F(M) = d \\ = F\left(\underline{x_p+1}, \underline{y_p-\frac{1}{2}}\right)$$

$$= (x_p+1)^2 + (y_p - \frac{1}{2})^2 - R^2$$

Midpoint Circle



$$d = F(M)$$

$$d \geq 0 \Rightarrow$$

E

$$d < 0 \Rightarrow$$

E

new midpt. M_{new}

$$x = x + 1$$

$$y = y - 1$$

new midpt. M_{new}

$$x = x + 1$$

$$y = y$$

Midpoint Circle

Asumsi jika sudah diplot piksel pada titik (x_k, y_k) , selanjutnya diharuskan untuk memilih antara $(x_k + 1, y_k)$ atau $(x_k + 1, y_k - 1)$.

Decision variable dapat didefinisikan sebagai:

$$\begin{aligned} p_k &= f_{circ}(x_k + 1, y_k - \frac{1}{2}) \\ &= (x_k + 1)^2 + (y_k - \frac{1}{2})^2 - r^2 \end{aligned}$$

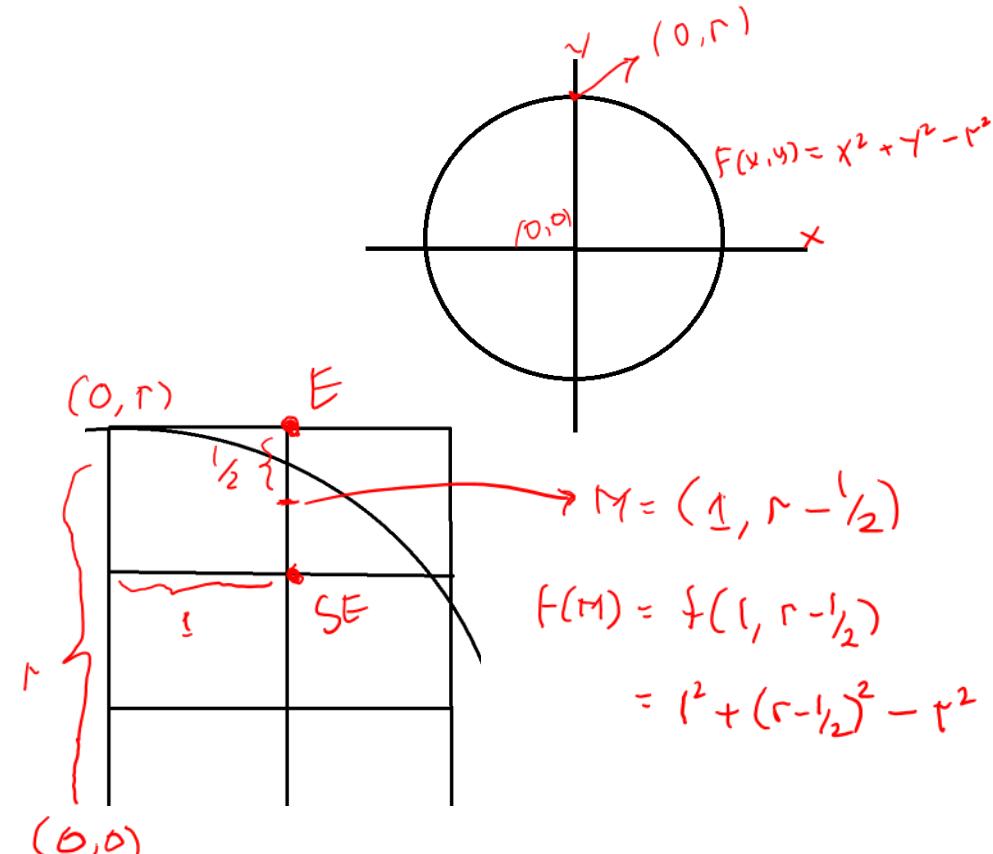
Jika $P_k < 0$ midpoint ada berada di dalam lingkaran dan titik y_k lebih dekat ke lingkaran (**titik pixel E**) maka nilai y_k tetap (tidak berubah)

Selainnya $P_k > 0$ midpoint ada berada di luar lingkaran dan titik $y_k - 1$ lebih dekat ke lingkaran (**titik pixel SE**), maka nilai y_k **decrement-1**

Midpoint Circle

Diketahui jika jari jari lingkaran adalah r merupakan jarak dari titik $(0,0)$ ke garis lingkaran.

Penggambaran lingkaran dimulai pada kuadran 2 di titik x_0, y_0 yaitu $(0, r)$, maka untuk titik tengah dari kandidat pixel berikutnya dengan x sebagai iterator akan bernilai $\left(1, r - \frac{1}{2}\right)$. Titik tersebut yang dievaluasi ke dalam fungsi lingkaran $f(x, y) = x^2 + y^2 - r^2$



Midpoint Circle

Dengan mengevaluasi titik M ke dalam fungsi lingkaran maka didapatkan parameter (decision variable) sebagai diskrimator misalkan P

Untuk parameter awal (init) dihasilkan:

$$P_0 = f_{circ}(M) = f_{circ}\left(1, r - \frac{1}{2}\right)$$

$$P_0 = 1 + \left(r - \frac{1}{2}\right)^2 - r^2$$

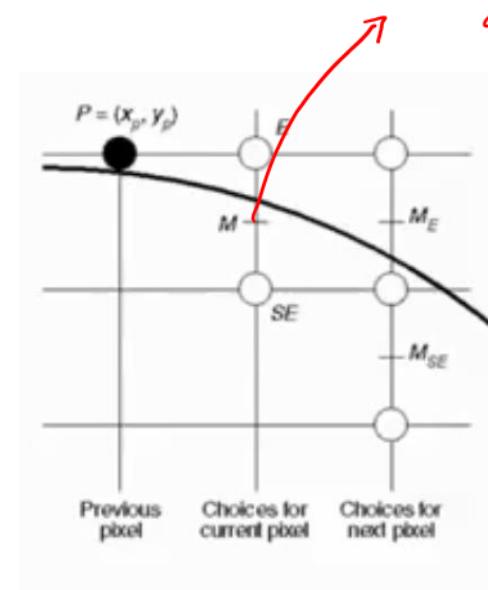
$$P_0 = \frac{5}{4} - r$$

$$P_0 \approx 1 - r \text{ (approximate)}$$

Jadi parameter awal untuk penggambaran lingkaran dihitung dengan nilai $1 - r$

$P_0 < 0 \rightarrow$ titik E
 (x_p+1, y_p)
atau

$P_0 > 1 \rightarrow$ titik SE
 (x_p+1, y_p-1)



Midpoint Circle

Kemudian jika $P_k < 0$ decision variable selanjutnya dengan memasukkan titik M_E pada fungsi:

$$f_{circ}(M_E) = f_{circ}\left(2, r - \frac{1}{2}\right)$$

didapatkan:

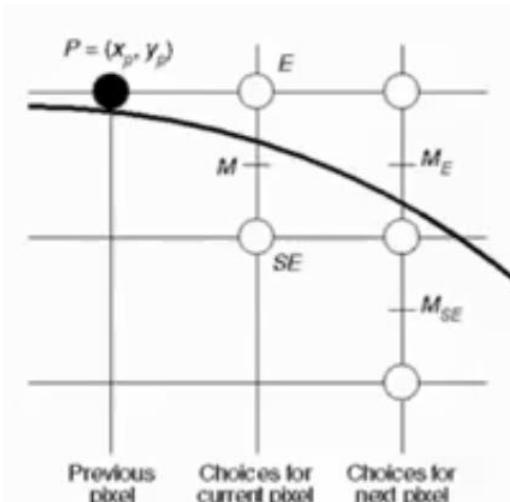
$$p_{k+1} = p_k + 2x_{k+1} + 1 \rightarrow p_k + 2x_k + 3$$

selainnya $P_k > 0$ memasukkan titik M_{SE} pada fungsi:

$$f_{circ}(M_{SE}) = f_{circ}\left(2, r - 1\frac{1}{2}\right)$$

didapatkan:

$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k-1} \rightarrow p_k + 2(x_k - y_k) + 5$$



Midpoint Circle (Algoritme)

Input radius r dan pusat lingkatan sebagai titik (x_0, y_0) , kemudian tentukan sebagai koordinat untuk titik awal dari lingkaran yaitu: $(x_0, y_0) = (0, r)$

Hitung Parameter awal dari algoritme midpoint circle dengan formulasi: $P_0 \cong 1 - r$

Dimulai dari $k = 0$ pada setiap posisi x_k , lakukan uji terhadap parameter keputusan dengan melihat tanda positif dan negatif.

Jika $P_k < 0$, maka titik selanjutnya sepanjang lingkaran dengan pusat lingkaran $(0,0)$ adalah $(x_k + 1, y_k)$ dimana titik pixel yang dipilih memiliki label E. Kemudian update parameter untuk titik selanjutnya dengan formulasi:

$$P_{k+1} = P_k + 2x_k + 3$$

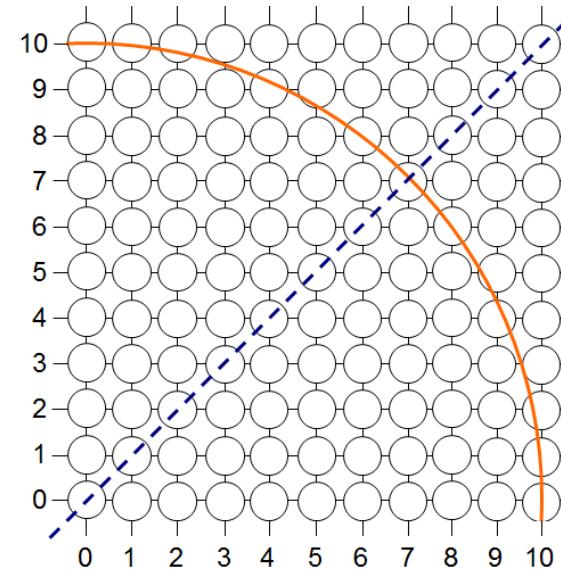
selainnya, maka titik selanjutnya sepanjang lingkaran dengan pusat lingkaran $(0,0)$ adalah $(x_k + 1, y_k - 1)$ dimana titik pixel yang dipilih memiliki label SE. Kemudian update parameter untuk titik selanjutnya dengan formulasi:

$$P_{k+1} = P_k + 2(x_k - y_k) + 5$$

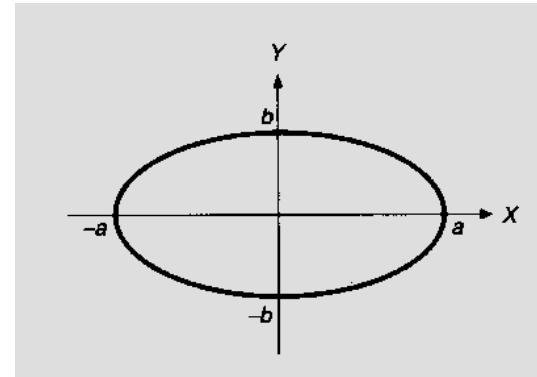
Ulangi sampai titik $x_k = y_n$

Latihan 1

Gunakan algoritme midpoint circle untuk menggambarkan lingkaran pada titik (0,0) dengan radius 10.



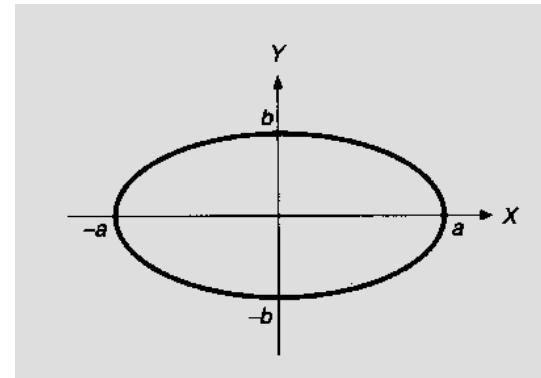
Scan Converting Ellipses



$$F(x, y) = b^2 x^2 + a^2 y^2 - a^2 b^2 = 0$$

- 2a is the length of the major axis along the x axis.
- 2b is the length of the minor axis along the y axis.
- The midpoint can also be applied to ellipses.
- For simplicity, we draw only the arc of the ellipse that lies in the first quadrant, the other three quadrants can be drawn by symmetry

Latihan 2



$$F(x, y) = b^2 x^2 + a^2 y^2 - a^2 b^2 = 0$$

Pahami Prosedur dari Bresenham's Midpoint ELLIPS Algorithm dan buat penggambaran elips dengan panjang $a= 4$, dan $b= 2$ (**sertakan referensi sumber yang digunakan**)

Jelaskan perbedaannya dengan Midpoint CIRCLE Algorithm



Terima Kasih

**Auzi Asfarian, M Asyhar Agmalaro,
Endang P Giri, Shervie N Neyman**

Departemen Ilmu Komputer, FMIPA
IPB University (Institut Pertanian Bogor), Bogor, Indonesia
asfarian@apps.ipb.ac.id | ux@apps.ipb.ac.id



**KOM1304 Grafika Komputer
dan Visualisasi**

Algoritme untuk Kurva non-linear & Transformasi

**Auzi Asfarian, M Asyhar Agmalaro,
Endang P Giri, Shervie N Neyman**

Departemen Ilmu Komputer, FMIPA
IPB University (Institut Pertanian Bogor), Bogor, Indonesia
asfarian@apps.ipb.ac.id | ux@apps.ipb.ac.id



IPB University
— Bogor Indonesia —

Departemen
Ilmu Komputer

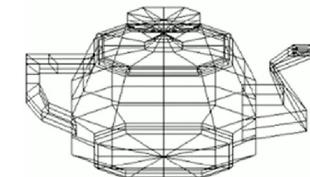
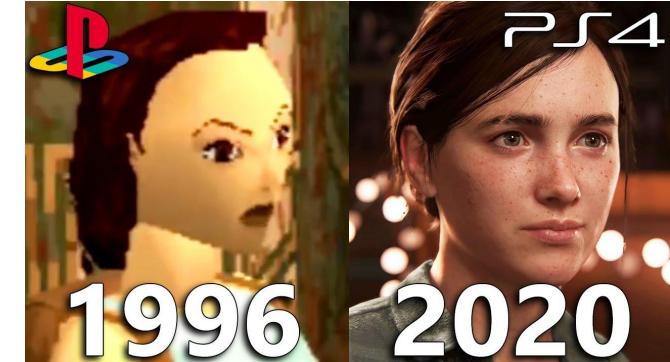
Bagian 1:

Kurva non-linear (Bazier function)

Kurva non-linear

Dalam penggambaran Objek grafis, agar objek semakin terlihat nyata maka diperlukan kurva yang tidak standar seperti garis saja, namun dibutuhkan kurva yang dapat melengkung non-garis/non-linear dengan lengkungan dapat disesuaikan berdasarkan deskripsi objek.

Ada banyak cara untuk menggambarkan kurva non-linear diantaranya dapat menggunakan Kurva Bazier, B-Spline dan lain-lain. Pada pembelajaran kali ini kita akan membahas mengenai kurva dengan fungsi Bazier



Preview (Polynomials)

Polynomial adalah penjumlahan dari variable yang dipangkatkan dan di multiplikasi dengan koefisien

$$\sum_{i=0}^n a_i x^n = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

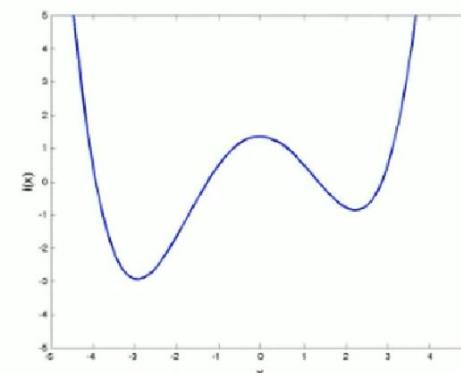
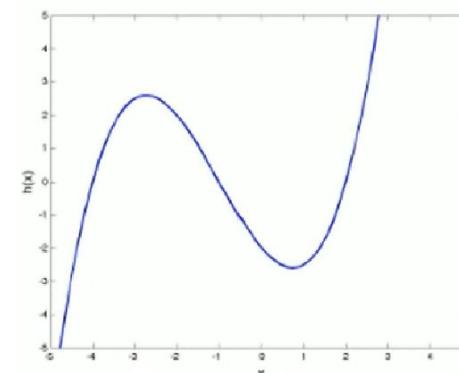
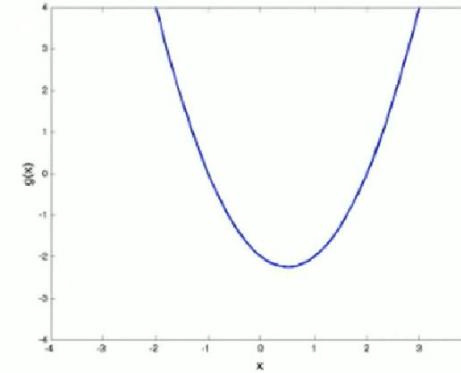
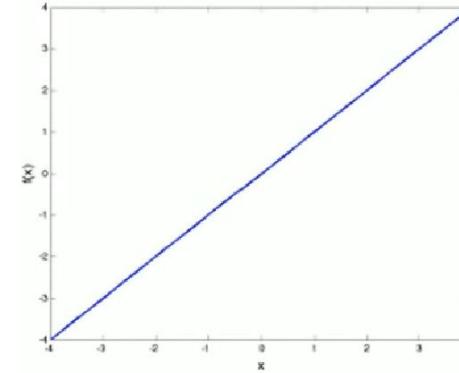
Derajat(order) dari polynomial adalah pangkat tertinggi dari variable

Polynomial dari derajat berbeda:

- Jika berderajat 1 disebut **linear**: $f(x) = 1 + x$
- Jika berderajat 2 disebut **quadratic**: $g(x) = 1 + x + x^2$
- Jika berderajat 3 disebut **cubic**: $h(x) = 1 + x + x^2 + x^3$
- Jika berderajat 4 disebut **quartic**: $i(x) = 1 + x + x^2 + x^3 + x^4$

Preview (Polynomials)

Semakin besar order/derajat dari suatu **Polynomial**, maka akan lebih banyak perubahan lengkungan (titik belok & titik balik) dari garis lurus untuk suatu kurva yang dapat dihasilkan



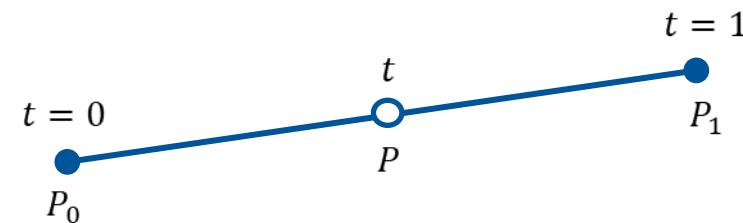
Persamaan Parametris

Kurva Bezier diekspresikan sebagai persamaan parametris, dimana terdapat parameter penentu t yang digunakan untuk menentukan nilai dari suatu variabel kurva, sebagai contoh:

$$\begin{aligned}x(t) &= (1 - t)x_0 + tx_1 \\y(t) &= (1 - t)y_0 + ty_1\end{aligned}$$

dimana $0 \leq t \leq 1$.

Dimisalkan $P_0 = (x_0, y_0)$, $P_1 = (x_1, y_1)$ dan $P = (x, y)$ maka: $P(t) = (1 - t)P_0 + tP_1$



Kurva Bezier

- **Kurva Bezier** adalah kurva parametris yang digunakan untuk menggambarkan garis dengan lengkungan (smooth)
- Didasarkan dari seorang engineer yang mendesain mobil Renault Pierre Bezier, kemudian mempopulerkan desain tersebut, namun algoritme dan kalkulasi dari kurva ini sebenarnya ditemukan oleh Paul de Casteljau 3 tahun sebelumnya
- Beberapa aplikasi yang menggunakan kurva Bezier CAD software, 3D modeling dan typefaces
- Sebuah derajat n dari kurva Bezier didefinisikan menggunakan **$n + 1$ control point**
- **Translasi dapat dengan mudah diaplikasikan ke control point**

Kurva Bezier

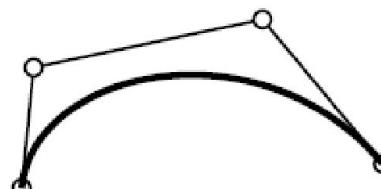
Persamaan garis Bezier akan membentuk titik-titik garis kurva sesuai control point yang didefinisikan



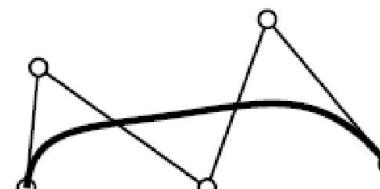
Degree 1



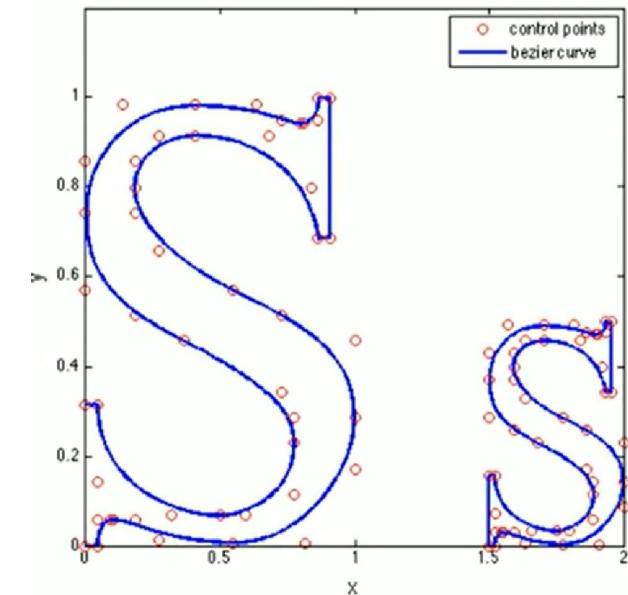
Degree 2



Degree 3

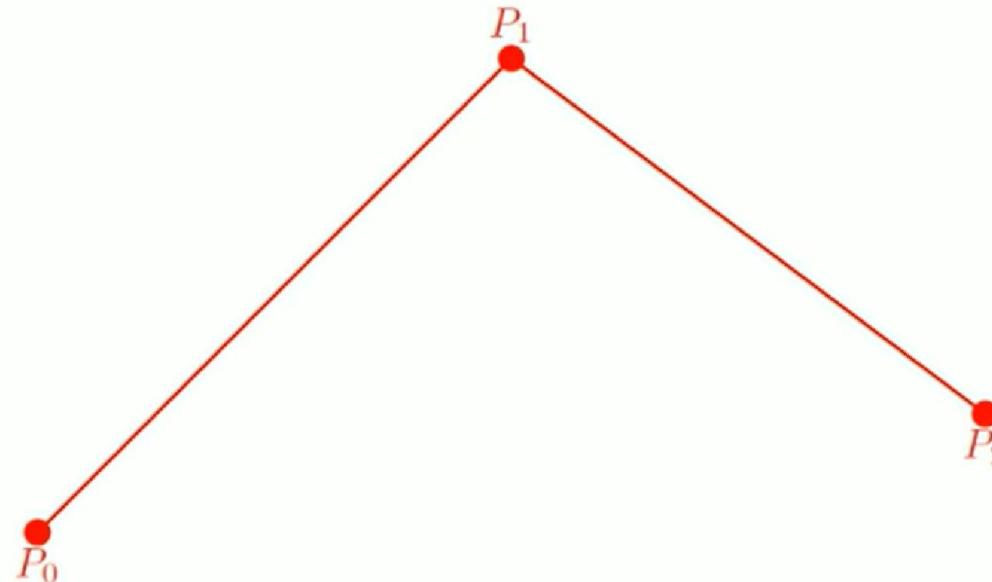


Degree 4



Kurva Bezier (Quadratic)

Q_0 dan Q_1 bergerak tepat di kurva dari $P_0 \rightarrow P_1$ dan $P_1 \rightarrow P_2$, sedangkan titik bezier berada tepat di kurva $Q_0 \rightarrow Q_1$



Kurva Bezier (Quadratic)

- Q_0 dan Q_1 adalah titik pada kurva $P_0 \rightarrow P_1$ dan $P_1 \rightarrow P_2$

$$Q_0 = (1 - t)P_0 + tP_1$$

$$Q_1 = (1 - t)P_1 + tP_2$$

- $C(t)$ adalah titik pada Kurva Bezier pada garis $Q_0 \rightarrow Q_1$:

$$C(t) = (1 - t)Q_0 + tQ_1$$

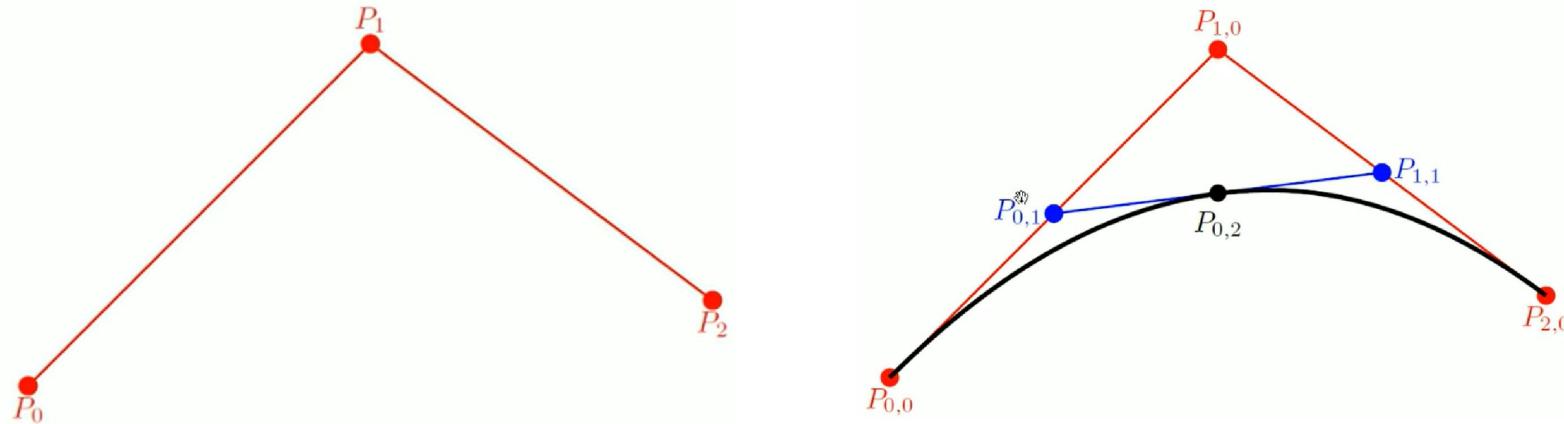
- Kombinasi dari persamaan itu akan didapatkan:

$$C(t) = (1 - t)^2 P_0 + 2t(1 - t)P_1 + t^2 P_2$$

- Proses penurunan persamaan di atas dikenal dengan **algoritme de Casteljau**

Kurva Bezier (Quadratic)

Algoritme de Casteljau, mendefinisikan control points dimana $P_{i,0}$ adalah original control points P_0 ke P_2 , $P_{i,1}$ adalah titik Q_0 ke Q_1 dan $P_{0,2}$ adalah $C(t)$

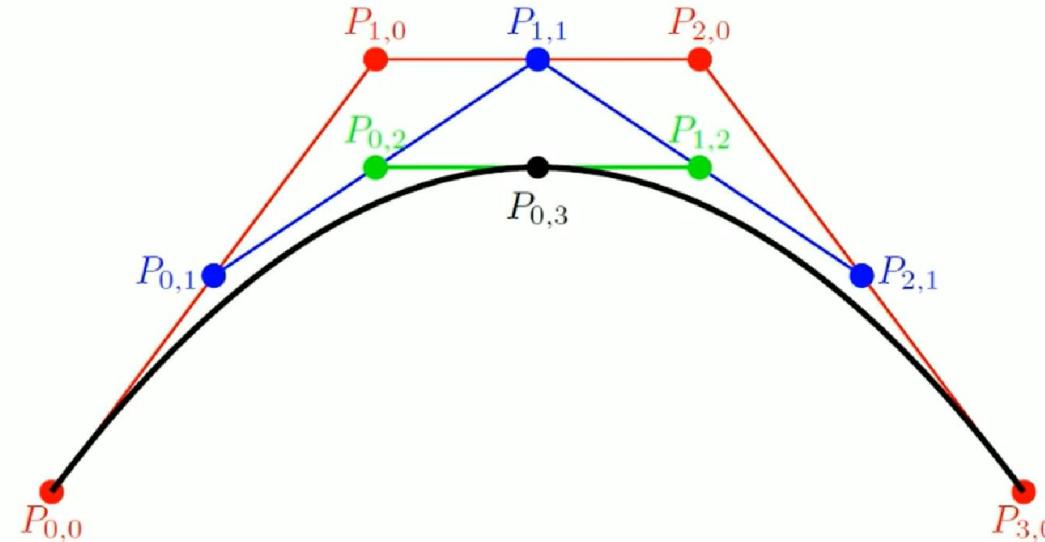


$$\text{secara umum yaitu: } C(t) = P_{0,2} = (1-t)^2 P_{0,0} + 2t(1-t)P_{1,0} + t^2 P_{2,0}$$

Kurva Bezier (Cubic)

Kurva Bezier Cubic didefinisikan dengan 4 control points: $P_{0,0}, P_{1,0}, P_{2,0}, P_{3,0}$

$$P_{0,3} = (1 - t)^3 P_{0,0} + 3t(1 - t)^2 P_{1,0} + 3t^2(1 - t)P_{2,0} + t^3 P_{3,0}$$



Kurva Bezier (General Form)

Kurva Bezier dalam bentuk umum didefinisikan dengan n control points P_i dimana $i = 0, 1, 2, \dots, n$

$$C(t) = \sum_{i=0}^n b_{i,n}(t)P_i$$

dimana $b_{i,n}(t)$ merupakan **Bernstein Polynomial**:

$$b_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

dan $\binom{n}{i}$ merupakan **kofisien binomial (pascal Triangle)** dengan notasi:

$$\binom{n}{i} = \frac{n!}{i! (n-i)!}$$

$n = 0:$	1			
$n = 1:$	1 1			
$n = 2:$	1 2 1			
$n = 3:$	1 3 3 1			
$n = 4:$	1	4	6	4 1

Kurva Bezier (General Form)

Sebagai contoh Kurva Bezier Cubic dalam bentuk umum:

$$b_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i},$$

$$b_{0,3}(t) = \binom{3}{0} t^0 (1-t)^{3-0} = (1-t)^3,$$

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}$$

$$b_{1,3}(t) = \binom{3}{1} t^1 (1-t)^{3-1} = 3t(1-t)^2,$$

$$n = 0: \quad \quad \quad 1$$

$$b_{2,3}(t) = \binom{3}{2} t^2 (1-t)^{3-2} = 3t^2(1-t),$$

$$n = 1: \quad \quad \quad 1 \quad \quad 1$$
$$n = 2: \quad \quad \quad 1 \quad \quad 2 \quad \quad 1$$

$$b_{3,3}(t) = \binom{3}{3} t^3 (1-t)^{3-3} = t^3,$$

$$n = 3: \quad 1 \quad \quad 3 \quad \quad 3 \quad \quad 1$$
$$n = 4: \quad 1 \quad \quad 4 \quad \quad 6 \quad \quad 4 \quad \quad 1$$

$$C(t) = \sum_{i=0}^n b_{i,n}(t) P_i = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t) P_2 + t^3 P_3$$

Kurva Bezier (Representasi dalam matriks)

- **Kurva Bezier Quadratic** dan sudah dijabarkan:

$$C(t) = (t^2 - 2t + 1)P_0 + (-2t^2 + 2t)P_1 + t^2P_2$$

menjadi:

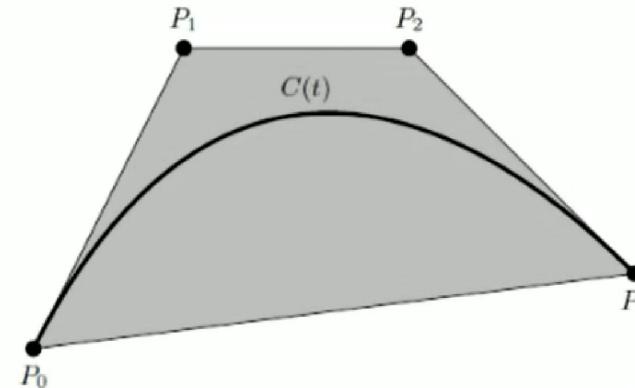
$$C(t) = \begin{pmatrix} P_0 & P_1 & P_2 \end{pmatrix} \begin{pmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} t^2 \\ t \\ 1 \end{pmatrix}$$

- **Kurva Bezier Cubic**

$$C(t) = \begin{pmatrix} P_0 & P_1 & P_2 & P_3 \end{pmatrix} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} t^3 \\ t^2 \\ t \\ 1 \end{pmatrix}$$

Properti Kurve Bezier

- Kurva Bezier dimulai pada Titik P_0 dan diakhiri pada titik P_n control points
- Derajat pada Kurva ditentukan dari $(n - 1)$ control points
- Sebuah kurva Bezier dapat dipisahkan (*split*) ke dua bentuk kurva Bezier
- Kurva Bezier akan berada di dalam polygon Control Points. Hal ini disebut dengan property convex hull





Contoh Kurve Bezier Quadratic

- Diberikan suatu kurva bezier derajat 2 dengan control point, titik awal $(0,0)$, $(2,2)$, titik akhir $(5,5)$. Jika akan dilakukan plotting pada kurva sebanyak 10 langkah, tentukan titik yang ke-5.

$$C(t) = (1 - t)^2 P_0 + 2t(1 - t)P_1 + t^2 P_2$$

Control Points $P_0 = (0,0)$; $P_1 = (2,2)$; $P_2 = (5,5)$

Ukuran langkah $k = 10$, maka setiap langkah: $\frac{1-0}{10} = 0.1$; maka titik kelima $t = 0,4$

$$C(t = 0.4) = (1 - 0.4)^2(0,0) + 2(0.4)(1 - 0.4)(2,2) + 0.4^2(5,5)$$

$$C(t = 0.4) = (0,0) + (0.8)(0.6)(2,2) + (0.16)(5,5)$$

$$C(t = 0.4) = (0,0) + (0.8)(0.6)(2,2) + (0.8, 0.8) = (1.76, 1.76)$$

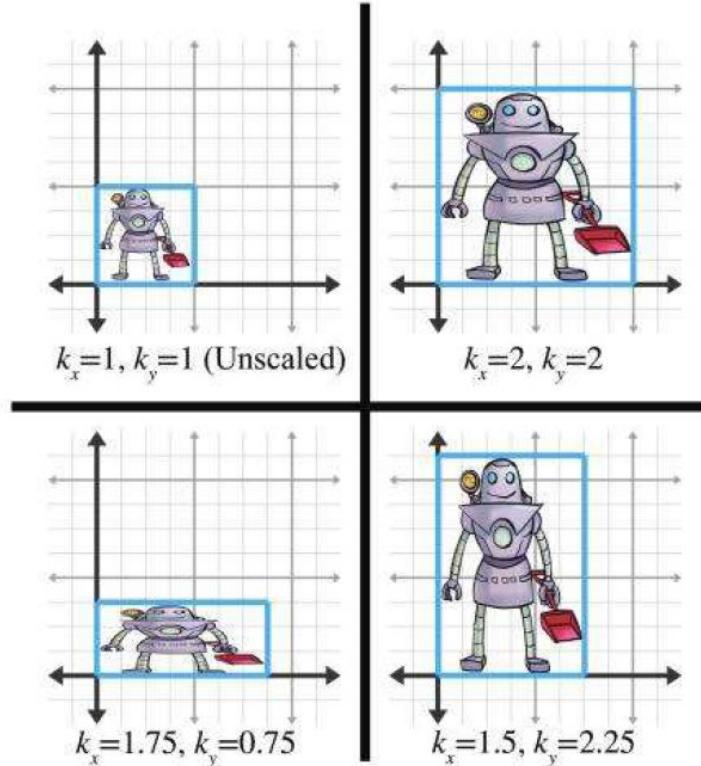


IPB University
— Bogor Indonesia —

Departemen
Ilmu Komputer

Bagian 2: Transformasi

Transformasi



Kenapa perlu ada Trasnformasi pada GKV?

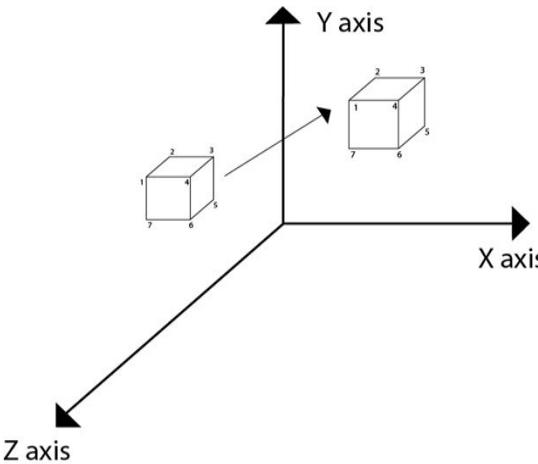
Transformasi diperlukan pada saat objek yang sudah digambarkan akan diletakan pada koordinat adegan, bersama dengan kumpulan-kumpulan objek lainnya.

Beberapa bentuk umum transformasi:

- Translasi
- Skalasi
- Rotasi

Transformasi objek direpresentasikan melalui **koordinat homogen/matriks homogen**, dan pada satu objek yang melibatkan banyak transformasi disebut sebagai **transformasi komposit**.

Transformasi



Transformasi ditujukan untuk menggambarkan operasi perubahan posisi, ukuran, ataupun view sebuah objek grafik

- o **Transformasi** yang digunakan untuk memperlihatkan pergeseran suatu objek dari satu posisi/kordinat ke posisi/koordinat lainnya disebut sebagai **Translasi**.
- o **Transformasi** yang memperlihatkan perubahan ukuran objek grafis, dengan mengalikan setiap titik terhadap suatu scalar disebut sebagai **Skalasi**.
- o **Transformasi** yang memperlihatkan perputaran objek terhadap titik origin dengan suatu sudut putaran tertentu (θ) disebut **Rotasi**.

Bentuk trasnformasi lainnya yang dimodifikasi dari bentuk dasar yaitu **refleksi** dan **shear**

Matriks Homogen Transformasi

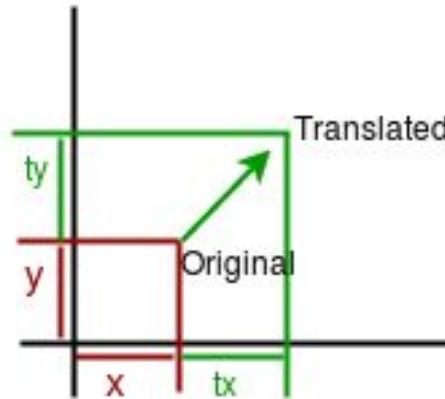
Semua transformasi dapat direpresentasikan sebagai matriks, dan untuk objek 2D menggunakan matriks berukuran 3×3

Menerapkan transformasi ke suatu titik/vertex dilakukan dengan mengalikan matriks tsb dengan vertex-nya

Seperti pada contoh penggambaran Garis yang memiliki vertex, maka setiap nilai vertex pada garis harus dikalikan dengan matriks homogen transformasi

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a * x + b * y + c * z \\ d * x + e * y + f * z \\ g * x + h * y + i * z \end{bmatrix}$$

Translasi



Perubahan posisi objek (titik) dari titik $P(x, y)$ ke **posisi tertentu** $P'(x', y')$ dikarenakan penjumlahan nilai titik dengan suatu nilai scalar t_x untuk koordinat-x dan t_y untuk koordinat-y:

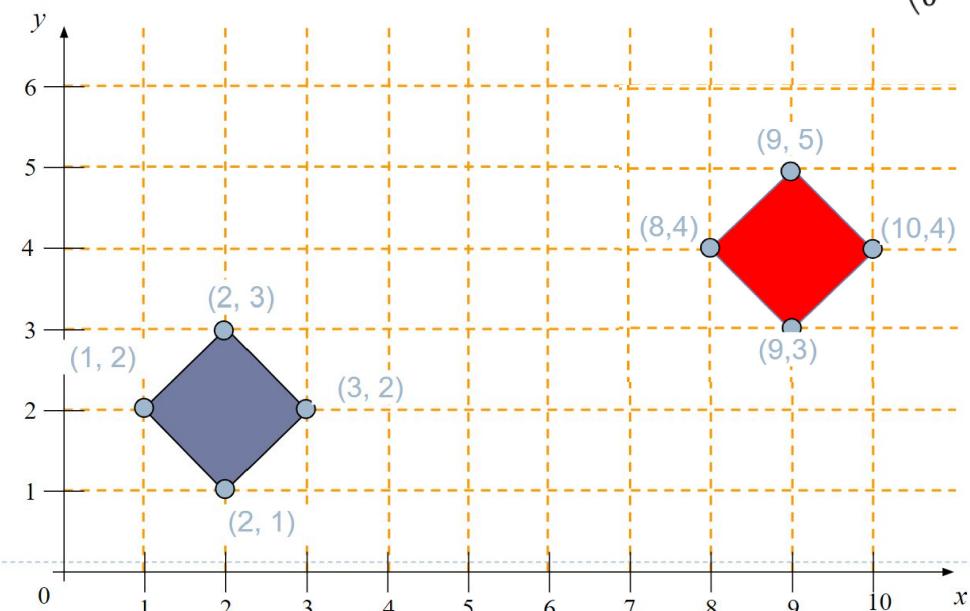
$$x' = x + t_x \text{ dan } y' = y + t_y$$

$P' = \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \end{pmatrix} = P + T$, dengan T merupakan representasi matriks homogen translasi

$$\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} (1 \times x) + (0 \times y) + (t_x \times 1) \\ (0 \times x) + (1 \times y) + (t_y \times 1) \\ (0 \times x) + (0 \times y) + (1 \times 1) \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \\ 1 \end{pmatrix}$$

Contoh Translasi

Translate the shape below by $(7, 2)$



$$\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} (1 \times x) + (0 \times y) + (t_x \times 1) \\ (0 \times x) + (1 \times y) + (t_y \times 1) \\ (0 \times x) + (0 \times y) + (1 \times 1) \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \\ 1 \end{pmatrix}$$

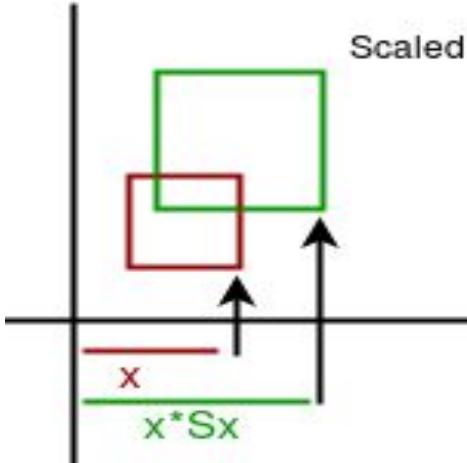
$$\begin{bmatrix} 1 & 0 & 7 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2+7 \\ 1+2 \\ 1 \end{bmatrix} = \begin{bmatrix} 9 \\ 3 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 7 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 2+7 \\ 3+2 \\ 1 \end{bmatrix} = \begin{bmatrix} 9 \\ 5 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 7 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 3+7 \\ 2+2 \\ 1 \end{bmatrix} = \begin{bmatrix} 10 \\ 4 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 7 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2+7 \\ 1+2 \\ 1 \end{bmatrix} = \begin{bmatrix} 9 \\ 3 \\ 1 \end{bmatrix}$$

Skalasi



Perubahan posisi objek (titik) dari titik $P(x, y)$ ke **posisi tertentu** $P'(x', y')$ dikarenakan perkalian nilai titik dengan suatu nilai scalar S_x untuk koordinat-x dan S_y untuk koordinat-y menyebabkan terjadinya perubahan **ukuran objek grafis**:

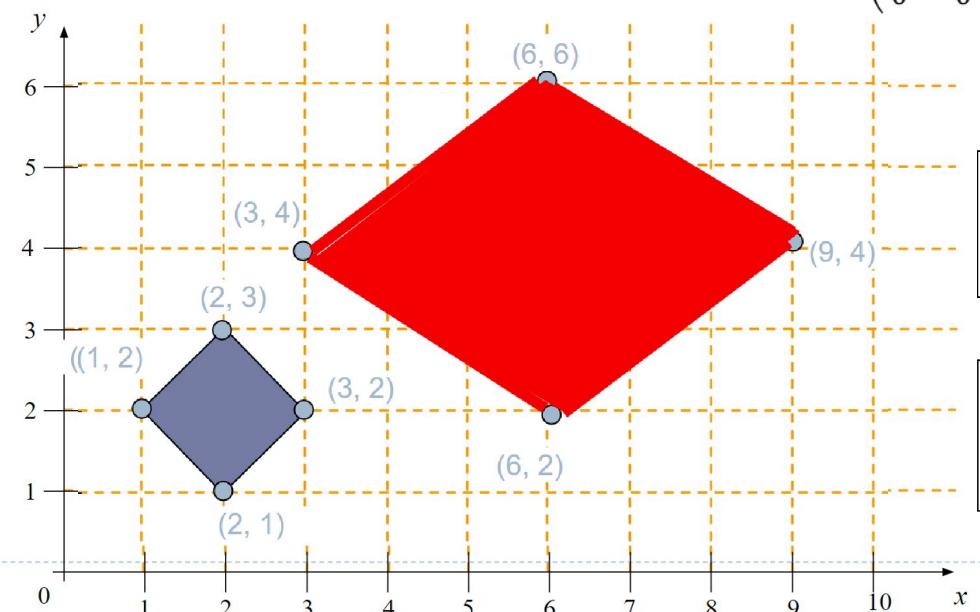
$$x' = S_x \cdot x \text{ dan } y' = S_y \cdot y$$

$P' = \begin{pmatrix} x' \\ y' \end{pmatrix} = \frac{S_x \cdot x}{S_y \cdot y} = S \cdot P$, dengan S merupakan representasi matriks homogen skalasi

$$\begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} (S_x \times x) + (0 \times y) + (0 \times 1) \\ (0 \times x) + (S_y \times y) + (0 \times 1) \\ (0 \times x) + (0 \times y) + (1 \times 1) \end{pmatrix} = \begin{pmatrix} S_x \cdot x \\ S_y \cdot y \\ 1 \end{pmatrix}$$

Contoh Skalasi

Scale the shape below by 3 in x and 2 in y



$$\begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} (S_x \times x) + (0 \times y) + (0 \times 1) \\ (0 \times x) + (S_y \times y) + (0 \times 1) \\ (0 \times x) + (0 \times y) + (1 \times 1) \end{pmatrix} = \begin{pmatrix} S_x \cdot x \\ S_y \cdot y \\ 1 \end{pmatrix}$$

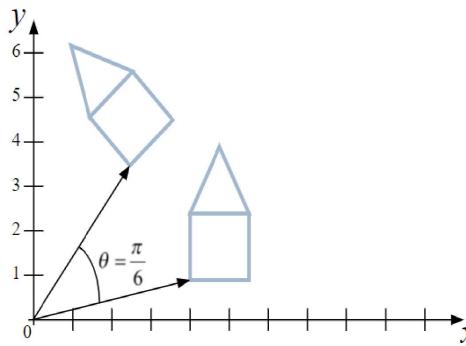
$$\begin{bmatrix} 1 & 0 & 7 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 3+7 \\ 2+2 \\ 1 \end{bmatrix} = \begin{bmatrix} 10 \\ 4 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 7 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2+7 \\ 1+2 \\ 1 \end{bmatrix} = \begin{bmatrix} 9 \\ 3 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 7 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 2+7 \\ 3+2 \\ 1 \end{bmatrix} = \begin{bmatrix} 9 \\ 5 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 7 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1+7 \\ 2+2 \\ 1 \end{bmatrix} = \begin{bmatrix} 8 \\ 4 \\ 1 \end{bmatrix}$$

Rotasi



Perubahan posisi objek (titik) dari titik $P(x, y)$ ke **posisi tertentu** $P'(x', y')$ dikarenakan perkalian nilai titik dengan suatu sudut putar tertentu θ pada titik origin $(0,0)$:

$$\begin{aligned}x' &= x \times \cos\theta - y \times \sin\theta \\y' &= x \times \sin\theta + y \times \cos\theta\end{aligned}$$

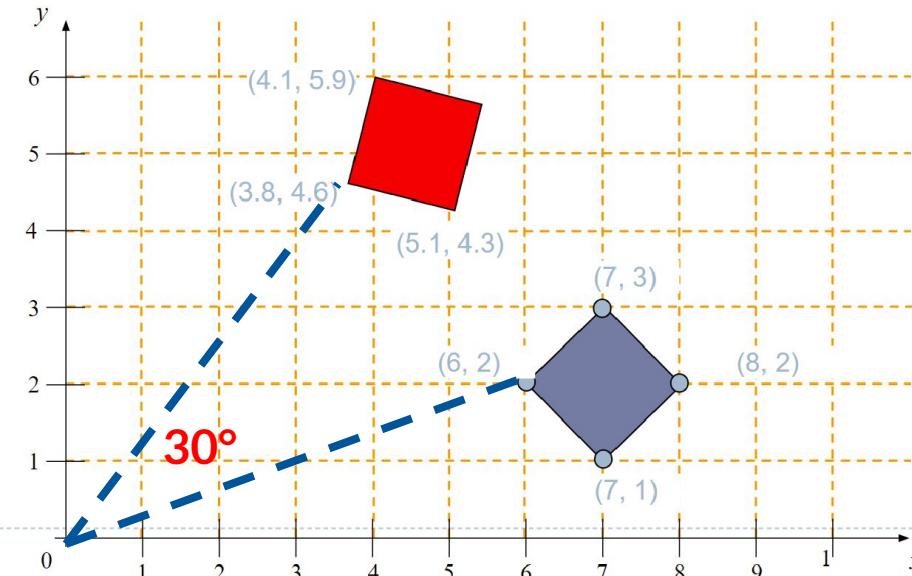
$$P' = \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \times \cos\theta - y \times \sin\theta \\ x \times \sin\theta + y \times \cos\theta \end{pmatrix} = P \cdot R,$$

dengan R merupakan representasi matriks homogen skalasi

$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} (\cos\theta \times x) + (-\sin\theta \times y) + (0 \times 1) \\ (\sin\theta \times x) + (\cos\theta \times y) + (0 \times 1) \\ (0 \times x) + (0 \times y) + (1 \times 1) \end{pmatrix} = \begin{pmatrix} x \times \cos\theta - y \times \sin\theta \\ x \times \sin\theta + y \times \cos\theta \\ 1 \end{pmatrix}$$

Contoh Rotasi

Rotate the shape below by 30° about the origin



$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x \times \cos\theta - y \times \sin\theta \\ x \times \sin\theta + y \times \cos\theta \\ 1 \end{pmatrix}$$

$$\begin{bmatrix} \cos 30 & -\sin 30 & 0 \\ \sin 30 & \cos 30 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 7 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.8 \times 7 - 0.5 \times 1 \\ 0.5 \times 7 + 0.8 \times 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 5.1 \\ 4.3 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} \cos 30 & -\sin 30 & 0 \\ \sin 30 & \cos 30 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 7 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.8 \times 7 - 0.5 \times 3 \\ 0.5 \times 7 + 0.8 \times 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 4.1 \\ 5.9 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} \cos 30 & -\sin 30 & 0 \\ \sin 30 & \cos 30 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 6 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.8 \times 6 - 0.5 \times 2 \\ 0.5 \times 6 + 0.8 \times 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 3.8 \\ 4.6 \\ 1 \end{bmatrix}$$

Inverse Transformation

Merupakan operasi untuk mengembalikan suatu hasil transformasi ke posisi sebelum transformasi.

$$T^{-1} = \begin{bmatrix} 1 & 0 & -dx \\ 0 & 1 & -dy \\ 0 & 0 & 1 \end{bmatrix}$$

$$R^{-1} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

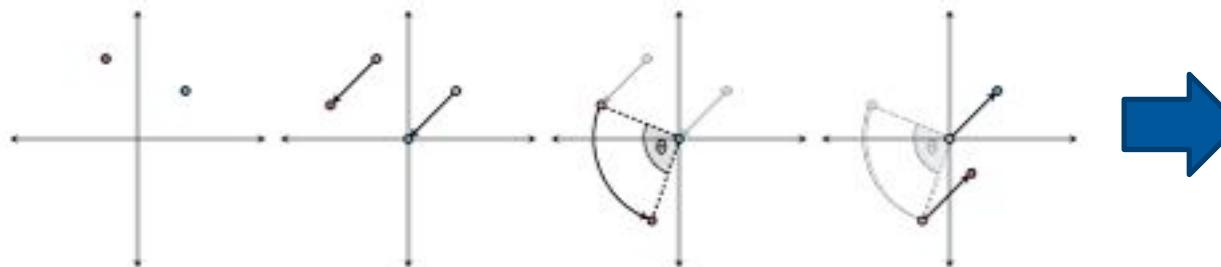
$$S^{-1} = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Combining Transformation (Rotation)

Rotasi di Titik yang bukan titik origin

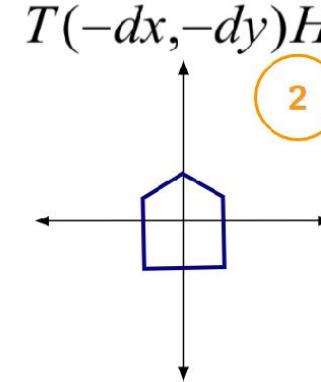
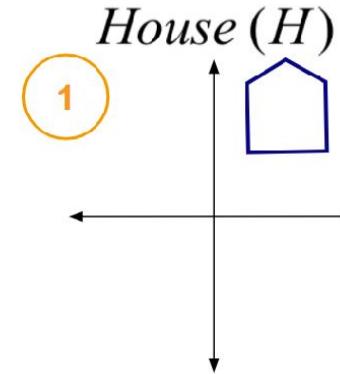


- Translasikan Titik ke Origin
- Rotasi dengan titik origin dengan sudut tertentu
- Translasikan kembali ke titik original (asal)



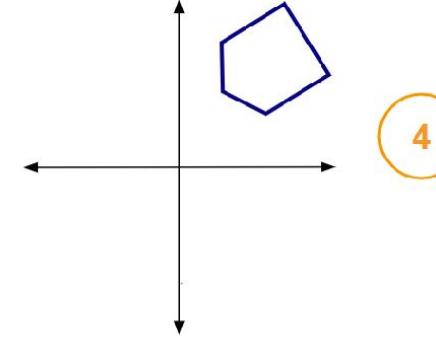
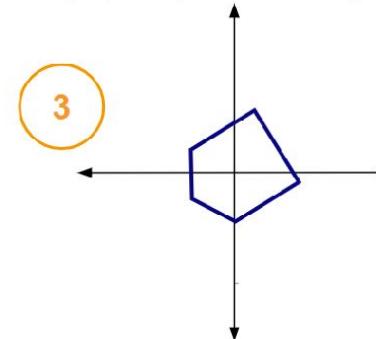
Transformasi
Komposit

Combining Transformation (Rotation)

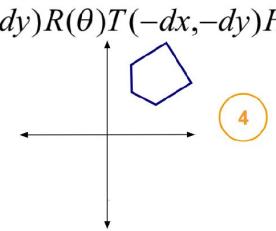
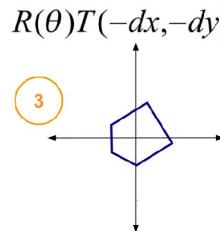
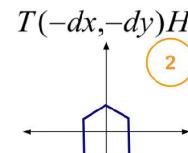
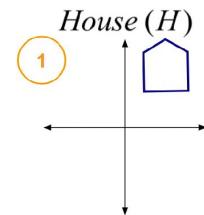


$$R(\theta)T(-dx, -dy)H$$

$$T(dx, dy)R(\theta)T(-dx, -dy)H$$



Combining Transformation (Rotation)



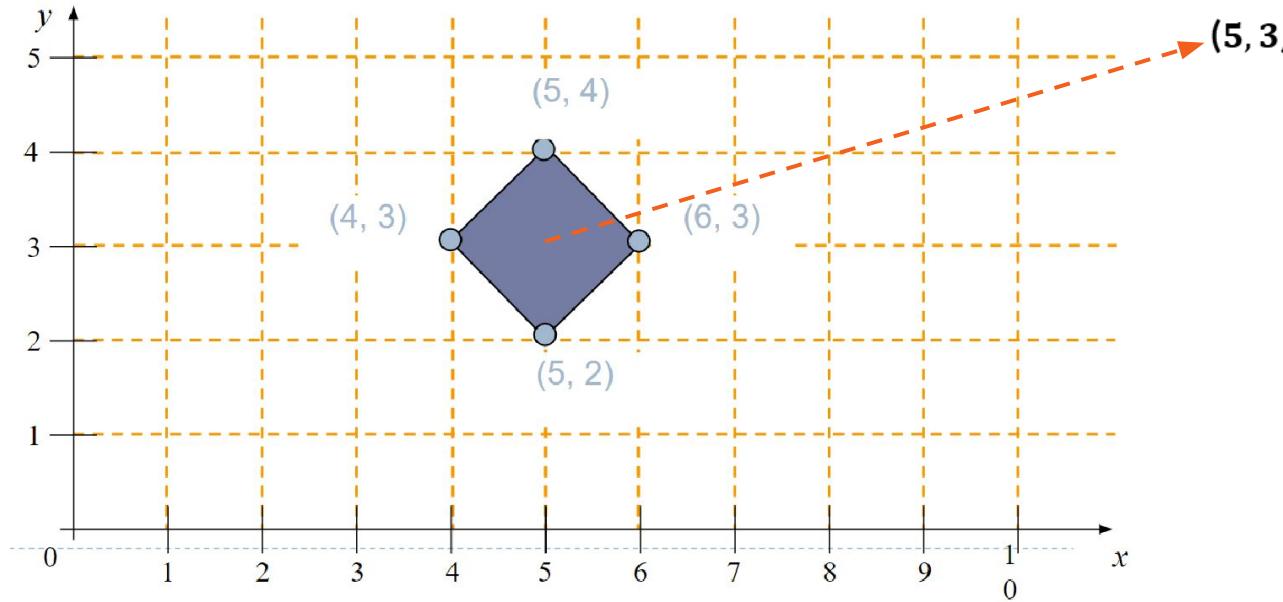
$$\begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -dx \\ 0 & 1 & -dy \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$v' = T(dx, dy)R(\theta)T(-dx, -dy)v$$

REMEMBER: Matrix multiplication is not commutative so order matters

Contoh Trasnformasi Komposit (Kasus rotasi)

Menggunakan perkalian matriks homogen, tentukan dan perlihatkan hasil rotasi dari objek berikut dengan sudut putar 45° , dan titik putar bukan di titik origin, melainkan titik $(5,3)$



Contoh Trasnformasi Komposit (Kasus rotasi)

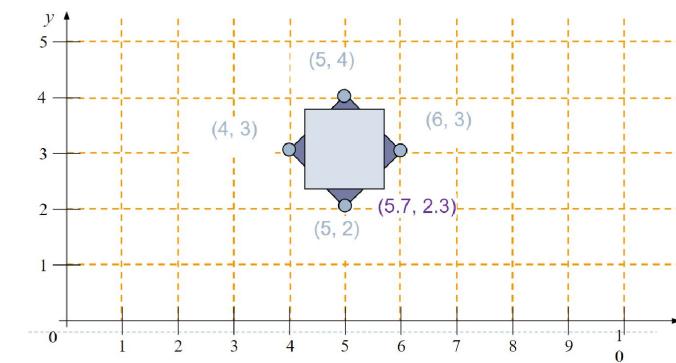
Menggunakan perkalian matriks homogen, tentukan dan perlihatkan hasil rotasi dari objek berikut dengan sudut putar 45° , dan titik putar bukan di titik origin, melainkan titik $(5,3)$

Jawaban:

$$\begin{bmatrix} 1 & 0 & 5 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \cos 45 & -\sin 45 & 0 \\ \sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -5 \\ 0 & 1 & -3 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 5 \\ 2 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 5 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0.7 & -0.7 & 0 \\ 0.7 & 0.7 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 5 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0.7 \\ -0.7 \\ 1 \end{bmatrix} = \begin{bmatrix} 5.7 \\ 2.3 \\ 1 \end{bmatrix}$$



Penyingkatan Kombinasi Transformasi

Hasil kombinasi dari transformasi (komposit) yang melibatkan titik P dan Matriks Homogen T, S, dan R dapat dipersingkat dalam memudahkan perhitungan. Hal ini disesuaikan dengan sifat-sifat Aljabar Matriks.

Berikut hasil penyingkatan Matriks Komposit/Matriks hasil kombinasi:

$$P' = (RT)P = RP + RT$$

$$P' = (RTR)P = R^2P + RT$$

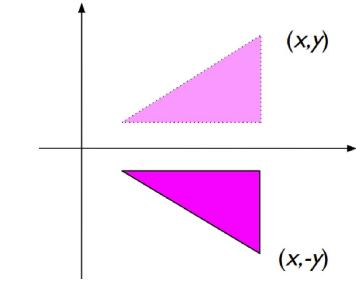
$$P' = (TRT^{-1})P = RP + R(-T) + T$$

Contoh Transformasi lainnya (refleksi)

Refleksi merupakan transformasi yang dimodifikasi dari bentuk umum transformasi yaitu skalasi.

Refleksi terhadap **koordinat sumbu-x** dan sama dengan **skalasi** dengan nilai S_x adalah **1** dan S_y adalah **-1**. Sehingga untuk refleksi terhadap sumbu-x dapat dinyatakan dengan:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$



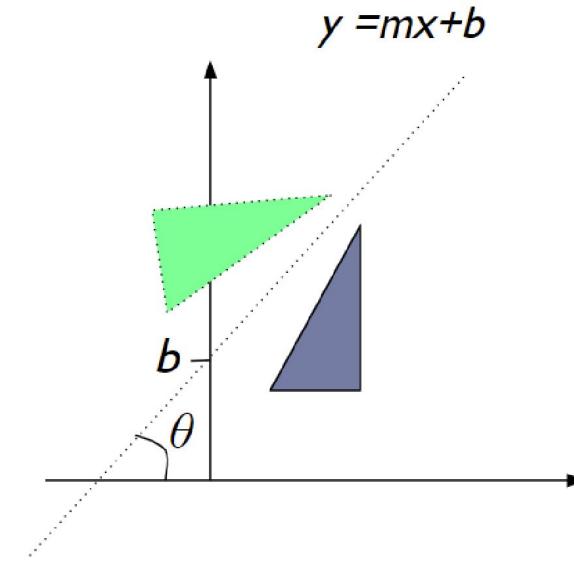
Refleksi terhadap **koordinat sumbu-y** dan sama dengan **skalasi** dengan nilai S_x adalah **-1** dan S_y adalah **1**. Sehingga untuk refleksi terhadap sumbu-x dapat dinyatakan dengan:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Refleksi dalam bentuk Transformasi Komposit

Refleksi sepanjang sebuah garis yang ditentukan dengan sebuah persamaan garis $y = mx + b$:

1. Translasikan titik $(0, -b)$ sehingga garis melewati titik origin
2. Rotasikan garis ke dalam sumbu-x dengan sudut $-\theta$
3. Refleksika dengan sumbu-x
4. Backward rotate (invers rotasi dari poin 2)
5. Backward translasi (invers translasi dari poin 1)



$$\theta = \tan^{-1}(m)$$



Terima Kasih

**Auzi Asfarian, M Asyhar Agmalaro,
Endang P Giri, Shervie N Neyman**

Departemen Ilmu Komputer, FMIPA
IPB University (Institut Pertanian Bogor), Bogor, Indonesia
asfarian@apps.ipb.ac.id | ux@apps.ipb.ac.id



**KOM1304 Grafika Komputer
dan Visualisasi**

3D - Objects Representations

**Auzi Asfarian, M Asyhar Agmalaro,
Endang P Giri, Shervie N Neyman**

Departemen Ilmu Komputer, FMIPA
IPB University (Institut Pertanian Bogor), Bogor, Indonesia
asfarian@apps.ipb.ac.id | ux@apps.ipb.ac.id



IPB University
— Bogor Indonesia —

Departemen
Ilmu Komputer

Bagian 1: 3D-Concepts

Learning Objectives

- 3D Concepts
- 3D Object Representations
 - Point cloud
 - Polyhedron
 - Quadrics
 - Spline curves representation
 - Blobby Object
 - Constructive Solid Geometry
 - Fractal & Particle System

3D Concept

- 3D Object : Representasi dari data geometrik 3 dimensi sebagai hasil pemrosesan dari efek depth (cahaya, texture, dll) pada objek berdimensi 2 (flat object)
- Pembuatan objek gambar dengan menggunakan 3 titik sebagai acuannya yaitu sumbu x, y dan z yang kemudian ditinjau secara matematis dalam melihat suatu objek, dimana gambar tersebut dapat terlihat secara menyeluruh dan nyata.

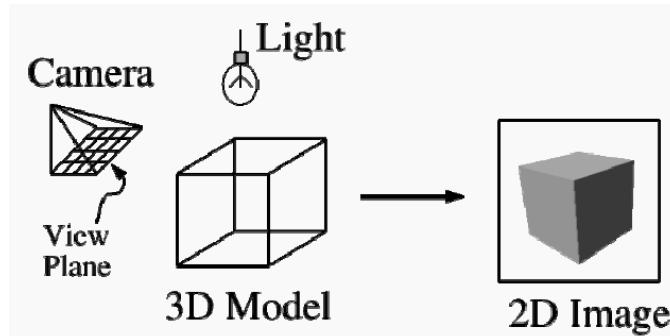
3D Concept



- **Graphics scenes** melibatkan berbagai jenis objek dan material surfaces
 - Kursi, Meja, Lampu, lukisan, Buku, Monitor
- Bagaimana merepresentasikan objek 3D ?

3D Concept

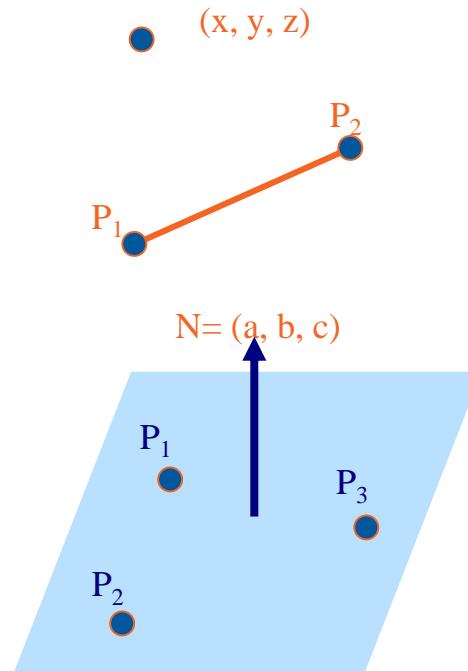
- Modeling : Merepresentasikan dimensi objek
- Rendering : Membentuk gambar 2D dari objek 3D
- Animation : Simulasi perubahan terhadap waktu



3D Concept

Primitif Geometri 3D

- Titik
 - Menandakan lokasi dalam dimensiruang
- Segmen garis
 - Kombinasi linear dua titik
- Bidang (plane)
 - Kombinasi linear tiga titik





IPB University
— Bogor Indonesia —

Departemen
Ilmu Komputer

Bagian 2:

3D Obj Representations

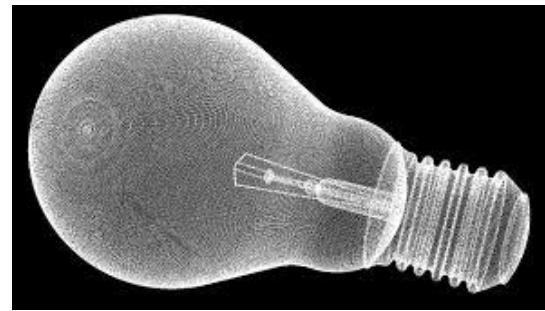
3D Object Representations

- Untuk merepresentasikan objek 3D, ada beberapa teknik
 - Membuat objek dalam representasi titik dalam ruang
 - Untuk membuat objek yang memiliki permukaan seperti polyhedrons ataupun ellipsoids maka menggunakan **polygon** dan **quadric**
 - Untuk membuat permukaan melengkung seperti pada sayap pesawat, gears, bodi mesin, etc, digunakan **Spline surfaces**
 - Untuk menyusun bentuk geometri dasar menjadi objek kompleks dapat menggunakan teknik **Constructive solid geometry**
 - Untuk memodelkan pegunungan, awan, tumbuhan, atau air terjun digunakan *procedural methods* seperti **fractals** dan **particle system**

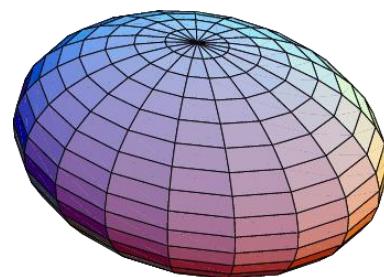
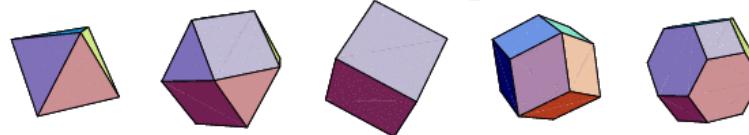
Teknik Representasi

- Dasar
 - Point cloud
 - Range image
- Permukaan
 - Polyhedron
 - Polygon mesh
 - Quadric
 - Bezier & spline
- Benda padat (solid)
 - Voxels
 - Sweep
 - CSG
- Advanced
 - Fraktal
 - Particle system
 - Physics-based

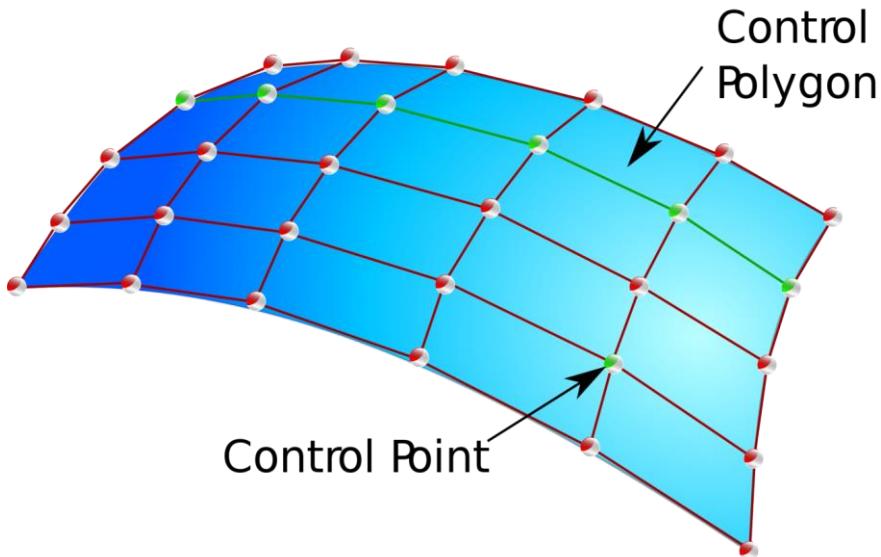
3D Object Representations



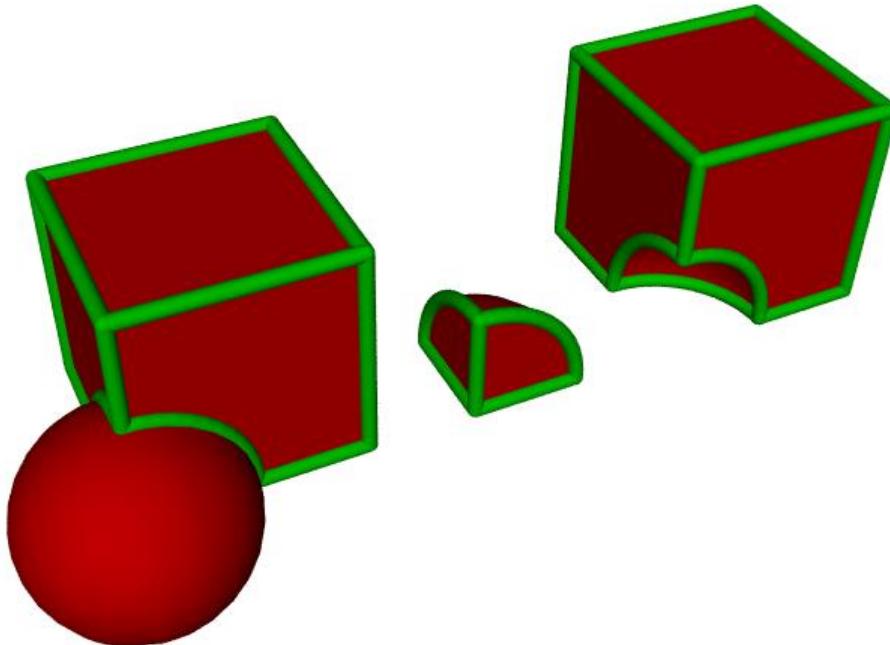
3D Object Representations



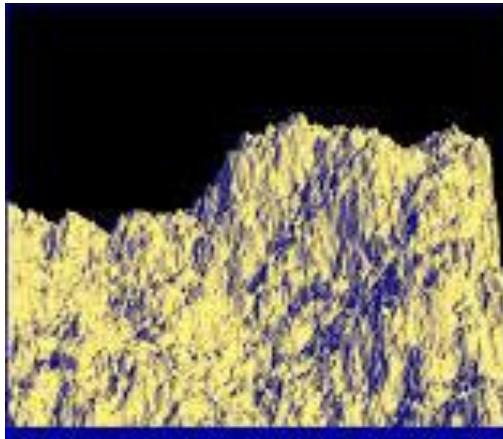
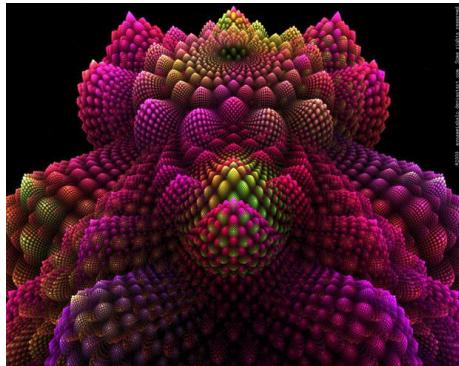
3D Object Representations



3D Object Representations

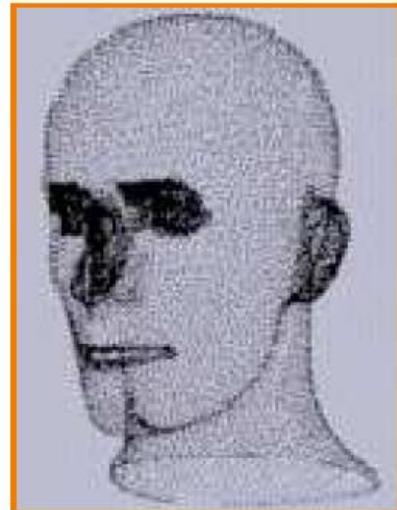


3D Object Representations



Point Cloud

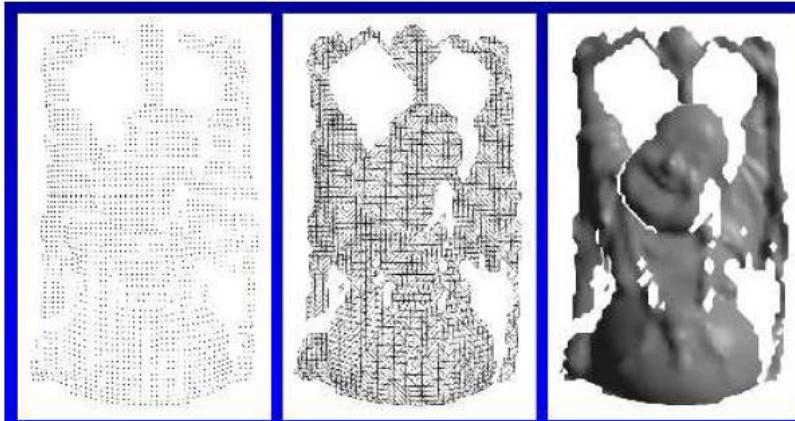
- Sekumpulan titik pada ruang 3D
 - Didapat dari range finder, computer vision, dll



Hoppe

Range Image

- Titik 3D yang menyatakan kedalaman



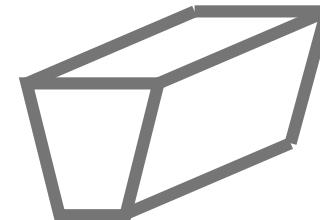
Range Image

Tesselation

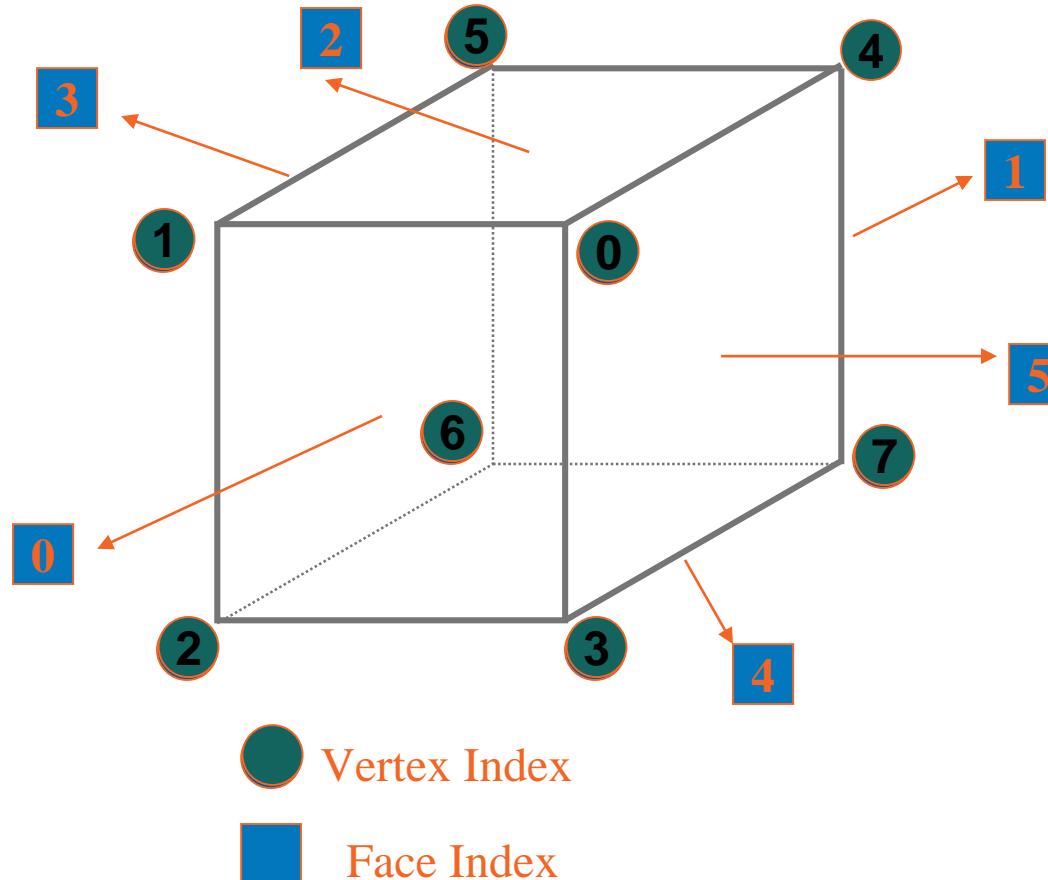
Range Surface

Polyhedron

- gabungan dari sejumlah **terhingga** (finite) daerah-daerah bidang-segi, sedemikian, sehingga setiap sisi dari suatu daerah bidang-segi merupakan sebuah sisi dari tepat sebuah bidang-segi lain.
- Polyhedron adalah rangkaian jala polygon (polygon mesh) dengan kriteria sbb
 - Sedikitnya 3 edge bertemu pada setiap vertex.
 - Setiap edge dipakai oleh 2 faces
 - Faces tidak saling menembus, tetapi berhenti pada suatu edge.
- Euler's formula : If F, E, V represent the number of faces, vertices and edges of a polyhedron, then
$$V + F - E = 2.$$

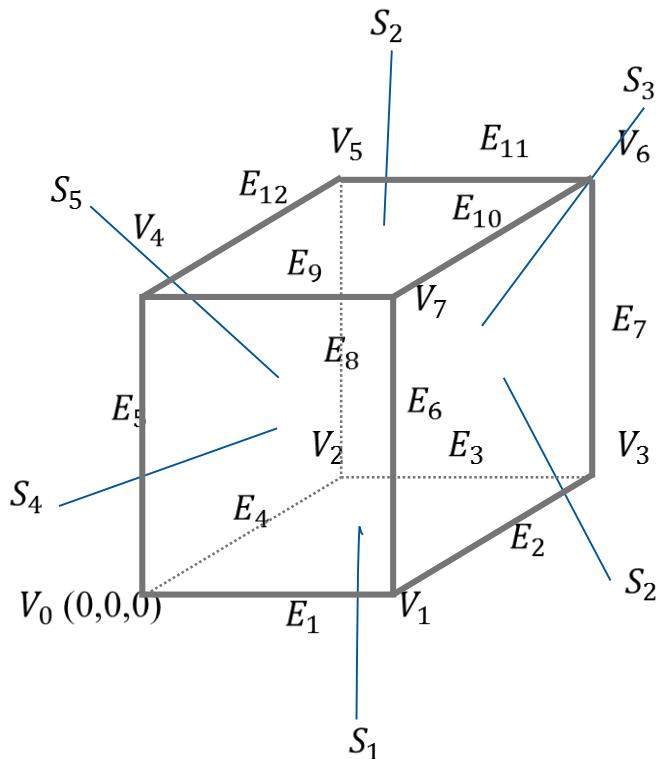


Vertices and Faces - E.g. Cube



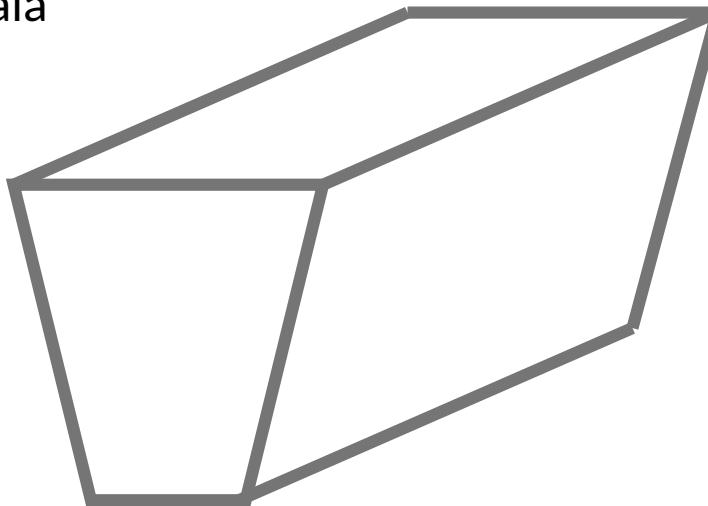
Contoh 1

- Jika diketahui sebuah objek 3D berbentuk kubus (polygon segi 4). Tentukan Vertex, edge dan surfacenya



Contoh 2

- Apakah objek berikut adalah polyhedron(jala polygon)?



Menggambar Polyhedron dalam Library OpenGL

- Ada 2 cara
 - Method1 : Fitting the surface with a [polygon mesh](#). Membungkus permukaan objek polyhedron dengan susunan jala polygon. Proses ini disebut juga dengan [surface tessellation](#)
 - Method 2 : Memakai fungsi yang disediakan library GLUT

Method-1 Polygon Mesh

- Dalam membuat polygon ke dalam bentuk *surfaces*, tidak terbatas hanya menggunakan GL_POLYGON
- Dapat juga menggunakan fungsi-fungsi berikut:
 - GL_TRIANGLES
 - GL_TRIANGLE_STRIP
 - GL_TRIANGLE_FAN
 - GL_QUADS
 - GL_QUAD_STRIP
- Bagaimana penggunaan GL_TRIANGLES untuk kasus pada contoh 1?

Draw Polygon Mesh in OpenGL (dengan vektor normal)

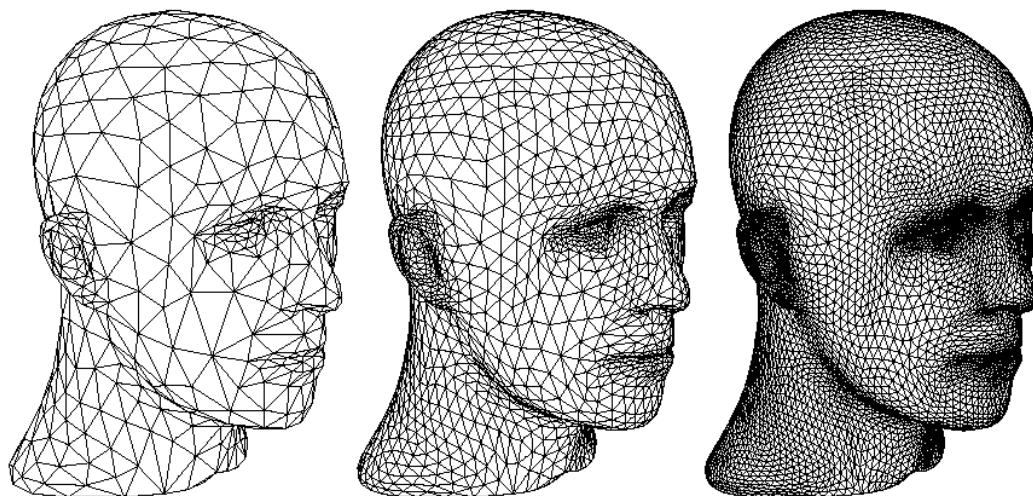
```
glBegin(GL_TRIANGLES); // draw a cube with 12 triangles polygon
// front surface =====
glVertex3fv(v0); // v0-v1-v7
glVertex3fv(v1);
glVertex3fv(v7);
glVertex3fv(v7); // v7-v4-v0
glVertex3fv(v4);
glVertex3fv(v0);
// right surface =====
glVertex3fv(v1); // v1-v3-v6
glVertex3fv(v3);
glVertex3fv(v6);
glVertex3fv(v6); // v6-v7-v1
glVertex3fv(v7);
glVertex3fv(v1);
// draw other 4 faces
glEnd();
```

Catatan:

Untuk mereduksi jumlah penganggilan fungsi dan penggunaan yang redundan, dapat menggunakan vertex arrays ke dalam 3 bentuk fungsi open GL yaitu: glDrawArrays(), glDrawElements() and glDrawRangeElements(). Atau pendekatan yang lebih baik menggunakan vertex buffer objects (VBO) or display lists

Polygon Mesh

- Polygon mesh ini juga bisa dipakai untuk memodelkan permukaan objek lainnya

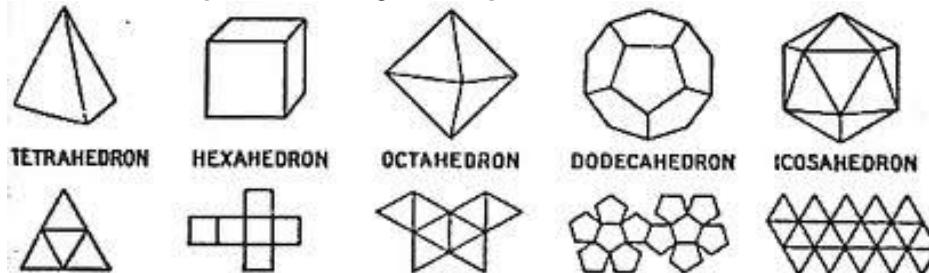


Method 2- OpenGL Polyhedron Functions

- Fungsi menghasilkan wire frames dimana dapat digunakan secara langsung dan mudah untuk *cube* yang dibentuk oleh regular polygon seperti: Tetrahedron, Octahedron, Hexahedron, Dodecahedron, atau Icosahedron
Contoh: glutWireX(), X merupakan nama cube
- Fungsi menghasilkan “polyhedra facets as shaded fill areas” - dengan karakteristik fungsi dideterminasi oleh properti lighting dan material
Ex: glutSolidX(), X merupakan nama cube

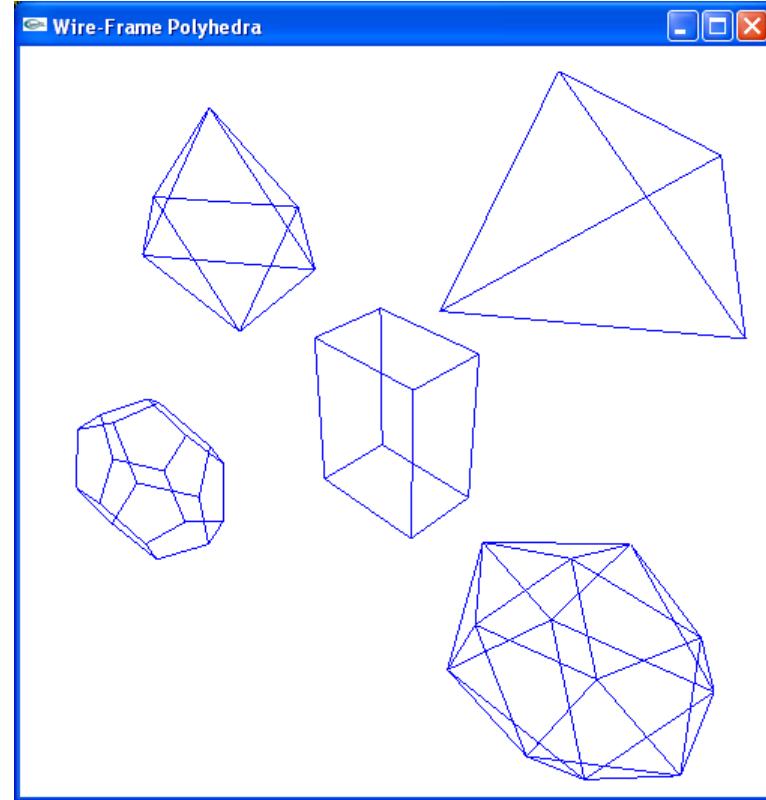
Regular Polyhedra (Platonic Solids)

- Jika semua face pada polyhedron adalah identik dan berupa regular polygon, maka polyhedron tsb disebut *platonic solid*.
- Hanya ada 5 jenis platonic solid

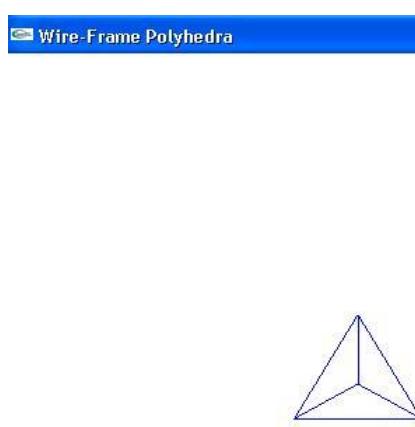


- Tetrahedron (or triangular pyramid) has 4 faces
- Hexahedron (or cube) with 6 faces
- Octahedron with 8 faces
- Dodecahedron with 12 faces
- Icosahedron with 20 faces

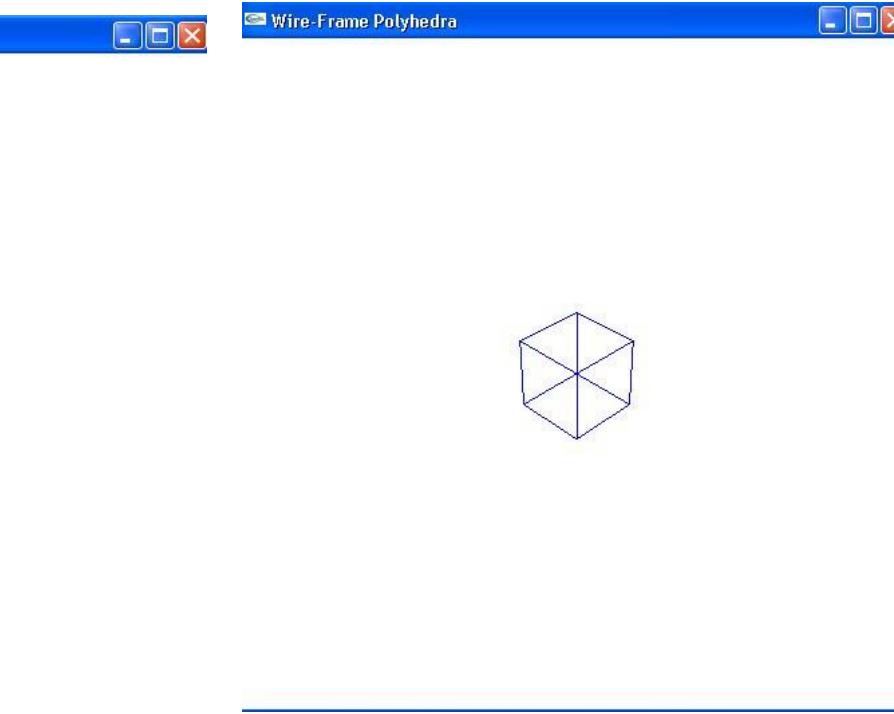
GLUT Library of Polyhedron Functions



glutWireTetrahedron() and glutWireCube(1.0)

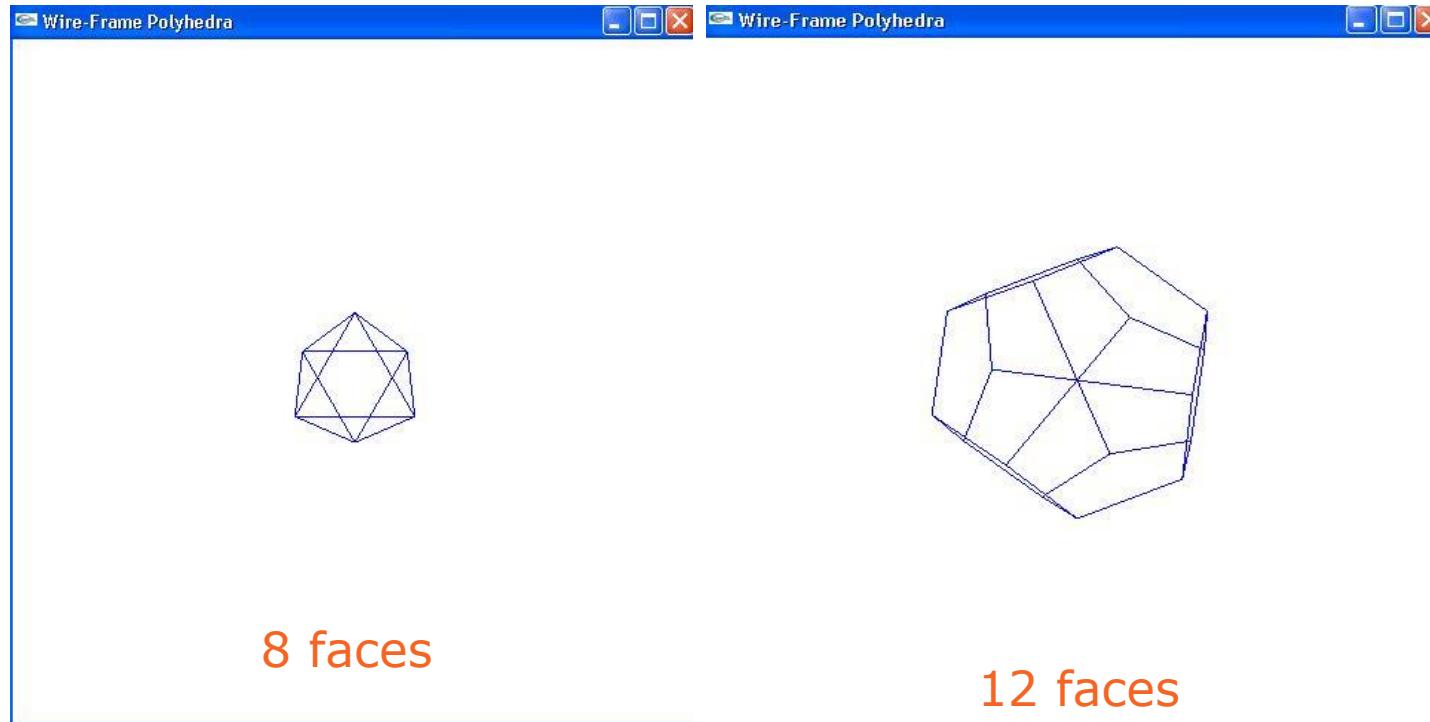


4 faces

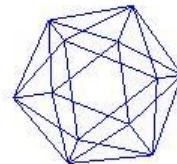


6 faces

glutWireOctahedron() and glutWireDodecahedron()



And, glutWireicosahedron()



20 faces

Quadratics

- Objek yang didefinisikan sebagai persamaan quadratics

- Sphere

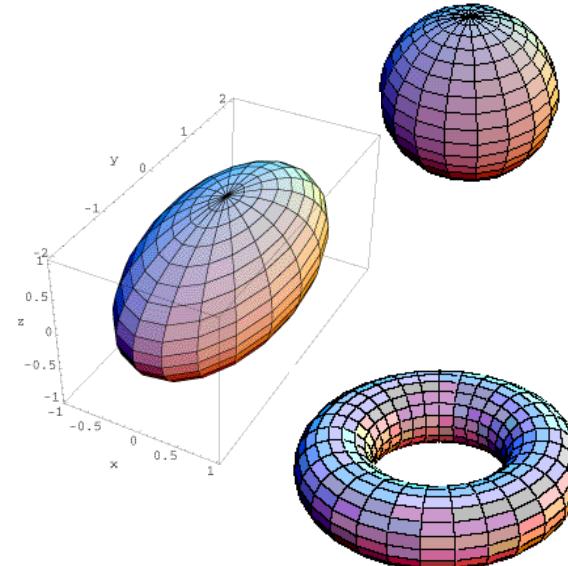
$$x^2 + y^2 + z^2 = r^2$$

- Ellipsoid

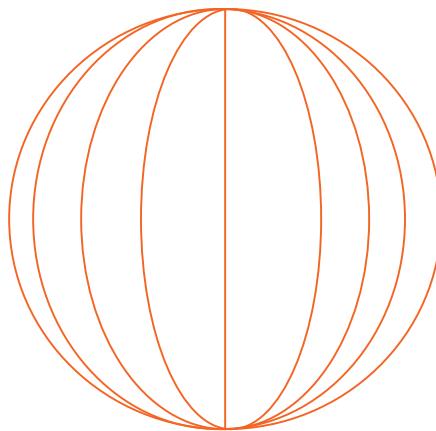
$$\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 = 1$$

- Torus

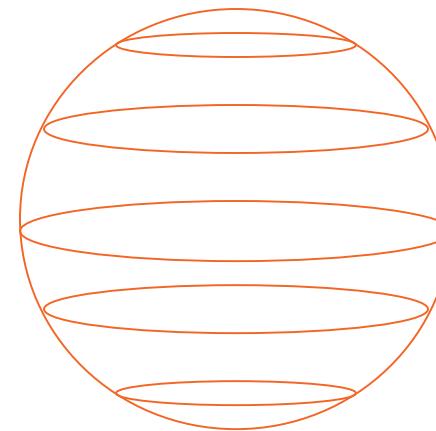
$$\left[r - \sqrt{\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2} \right]^2 + \left(\frac{z}{r_z}\right)^2 = 1$$



GLUT Sphere



Longitudes

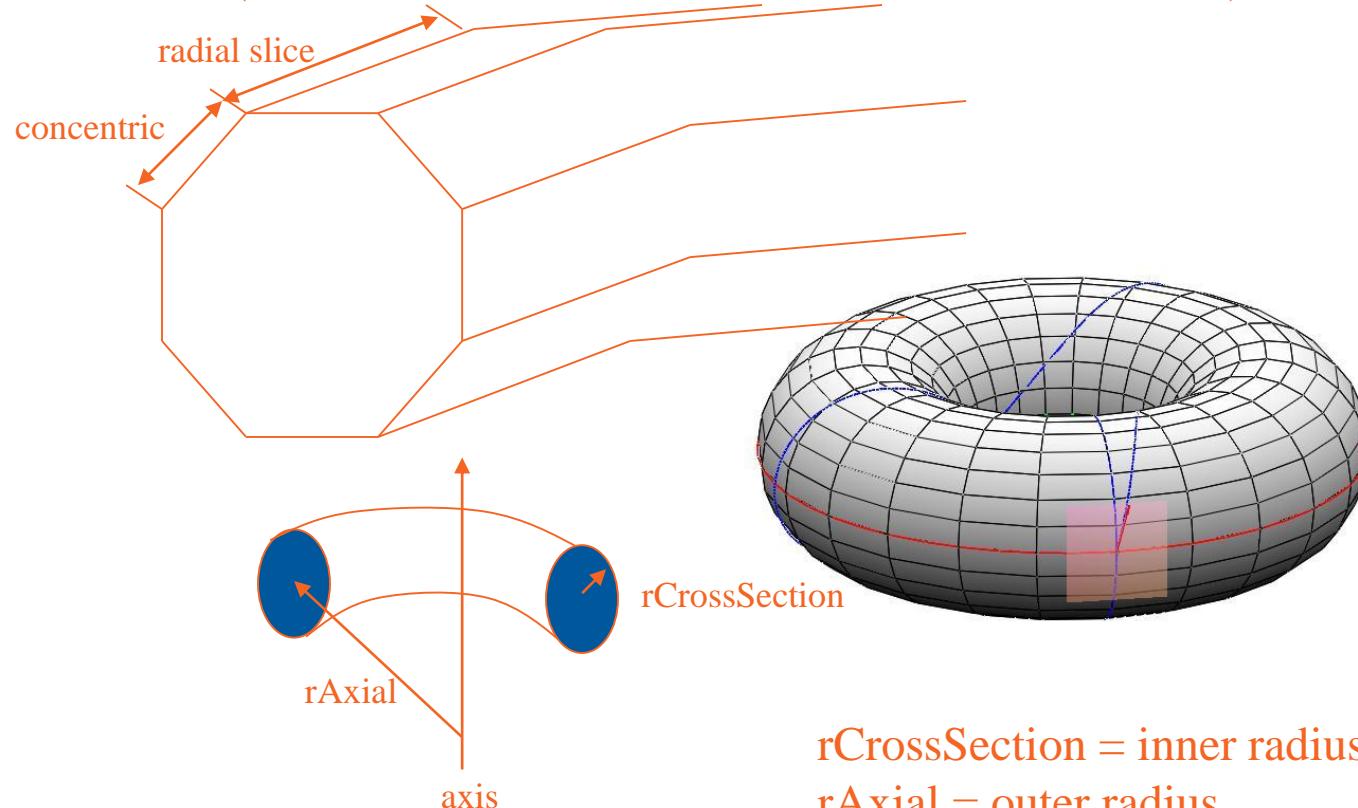


Latitudes

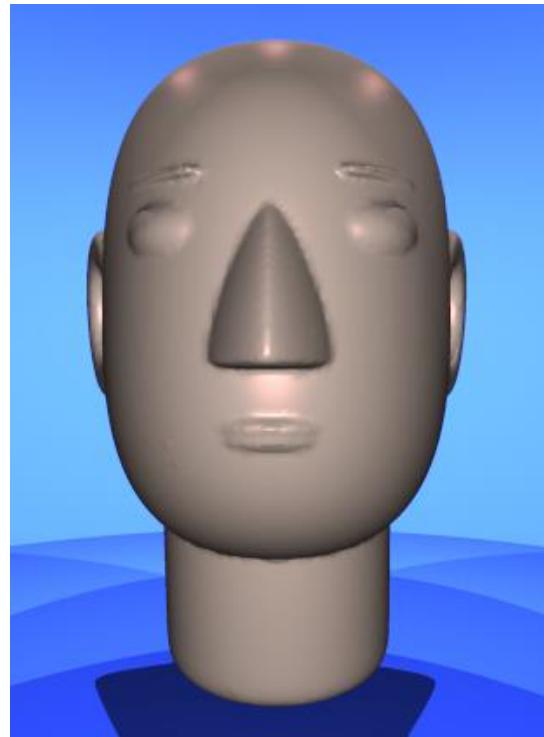
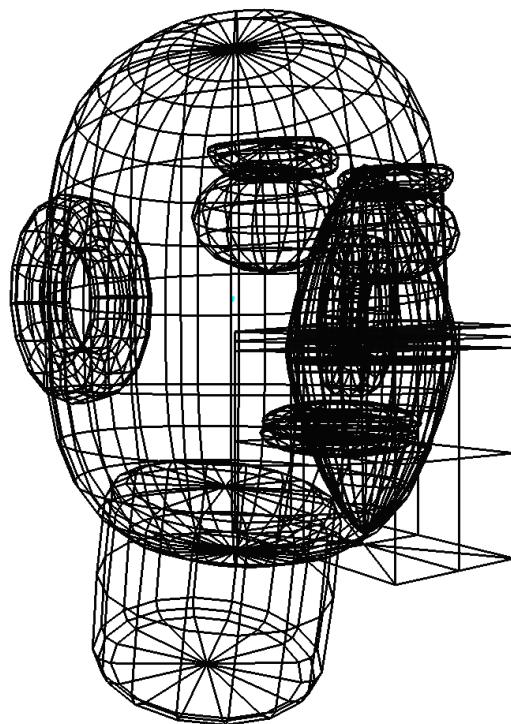
```
glutSolidSphere(r, nLongitudes, nLatitudes);
```

GLUT Torus

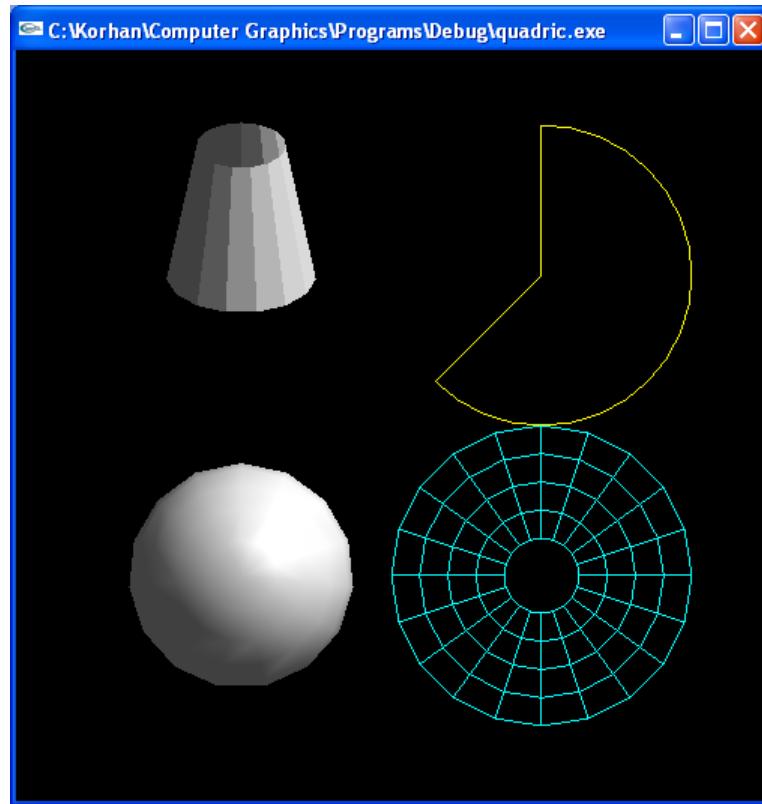
```
glutSolidTorus(rCrossSection, rAxial, nConcentrics, nRadialSlices);
```



Object with Quadrics

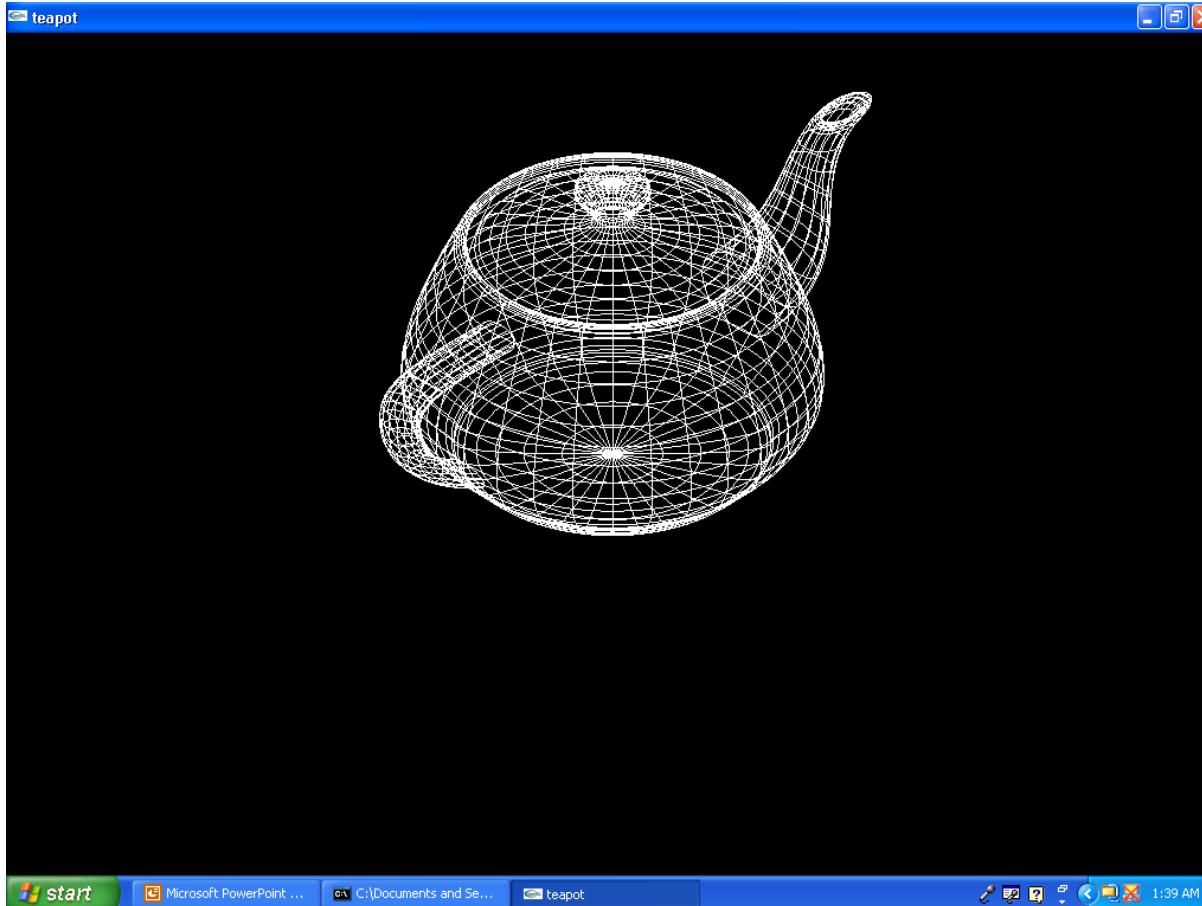


GLU Quadric-Surface Functions

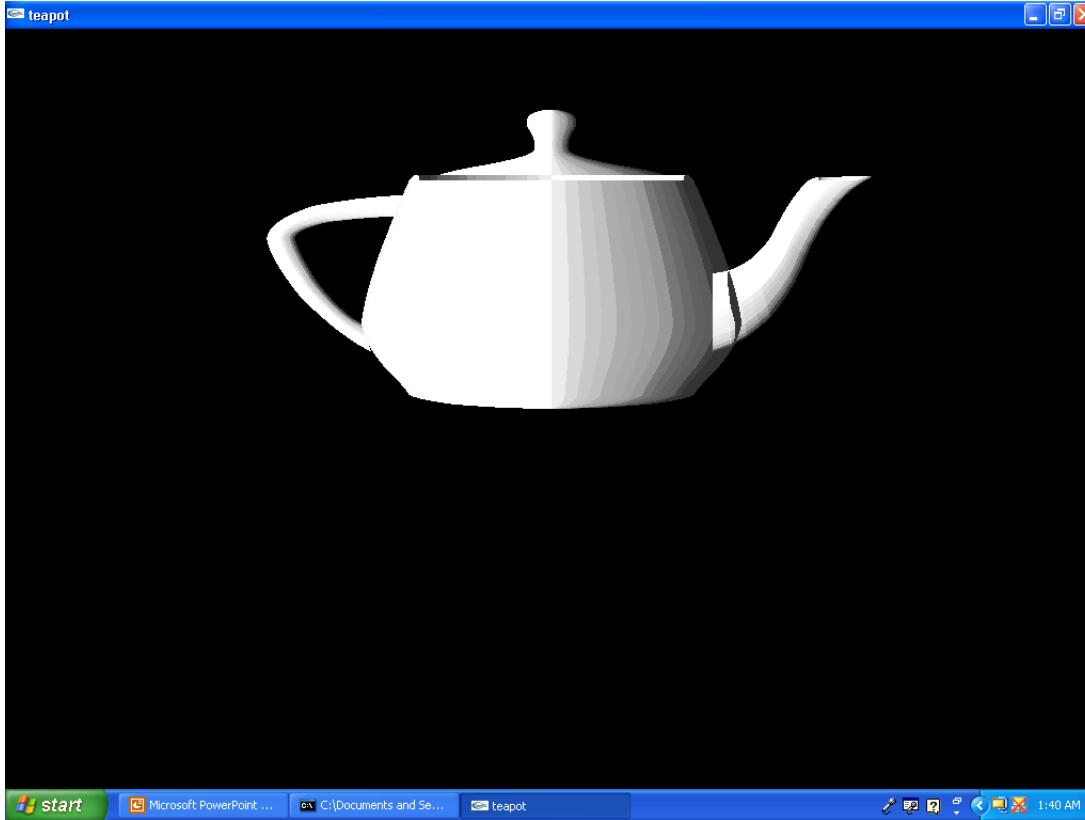


Quadric.c

wireframe



Lighting & shading



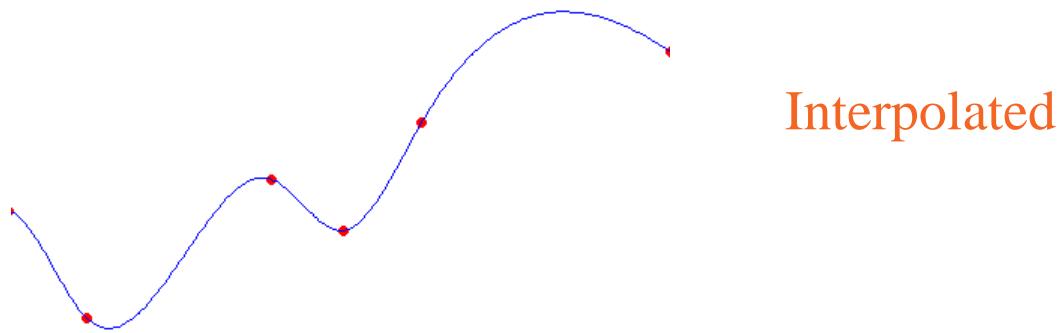
Texture mapped



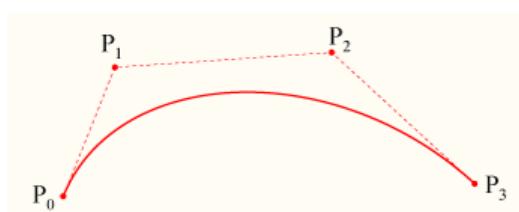
C. Spline Representations

- Splines curves digunakan untuk mendesain lengkungan pada permukaan berbasis pada sekumpulan (himpunan) titik pengontrol kurva
- Titik Pengontrol (Control points)
 - Himpunan titik koordinat yang mengontrol bentuk kurva
- Interpolasi (Interpolation)
 - Semua control points tersambung satu sama lain pada garis kurva

Spline Representations

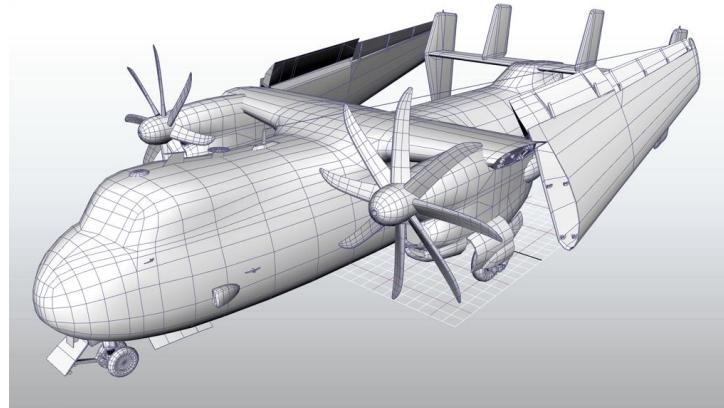
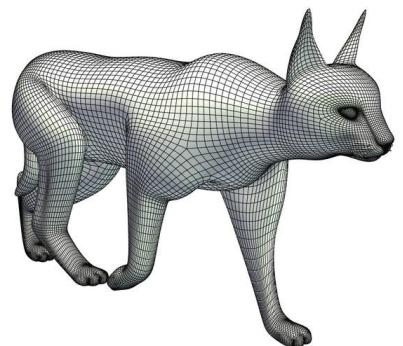
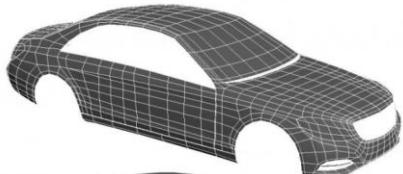


Interpolated



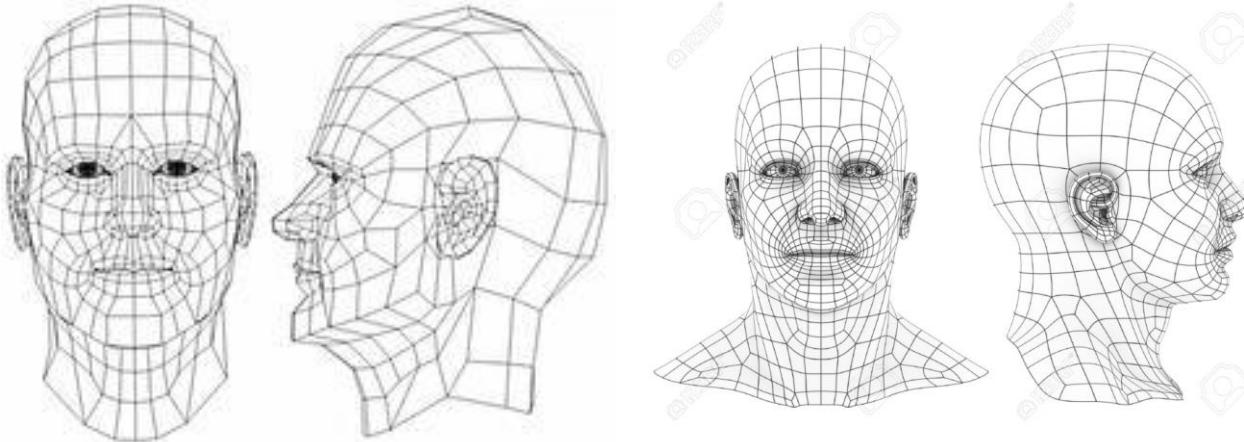
Approximate

Spline Representation



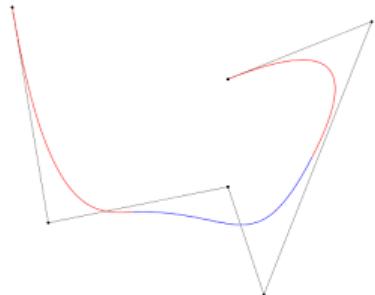
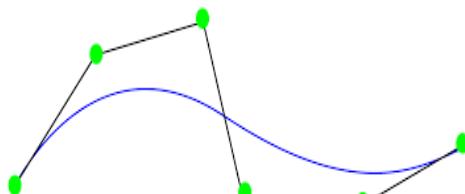
Spline Representation

- Perbedaan antara penggambaran Polygon dan Polygon with spline curve



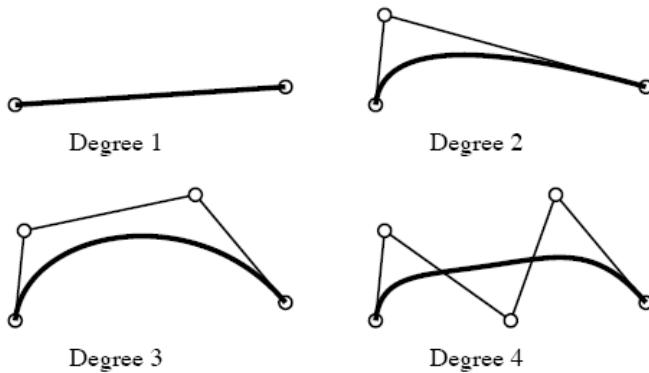
Bezier Spline Curves

- Developed by French engineer Pierre Bézier for use in the design of Renault automobile bodies
- Easy to implement
 - Widely used in CAD systems, graphics, drawing and painting packages



Bezier Curve Equations

- Diketahui sejumlah $n + 1$ control points,
 $p_k = (x_k, y_k, z_k)$
nilai k antara 0 sampai n
- Persamaan garis Bezier akan membentuk titik-titik garis kurva sesuai control point yang didefinisikan



Bezier Curve Equations

- Degree 1 - Linear Curve

$$\mathbf{B}(t) = \mathbf{P}_0 + t(\mathbf{P}_1 - \mathbf{P}_0) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1, t \in [0, 1]$$

- Degree 2

$$\mathbf{B}(t) = (1-t)^2\mathbf{P}_0 + 2(1-t)t\mathbf{P}_1 + t^2\mathbf{P}_2, t \in [0, 1].$$

- Degree 3

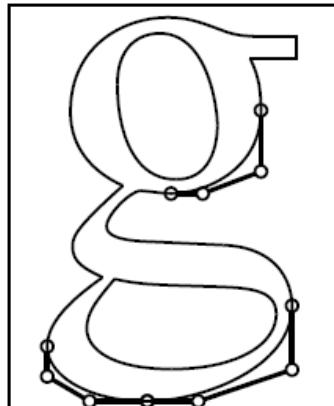
$$\mathbf{B}(t) = (1-t)^3\mathbf{P}_0 + 3(1-t)^2t\mathbf{P}_1 + 3(1-t)t^2\mathbf{P}_2 + t^3\mathbf{P}_3, t \in [0, 1].$$

- Degree n

$$\begin{aligned}\mathbf{B}(t) &= \sum_{i=0}^n \binom{n}{i} (1-t)^{n-i} t^i \mathbf{P}_i \\ &= (1-t)^n \mathbf{P}_0 + \binom{n}{1} (1-t)^{n-1} t \mathbf{P}_1 + \cdots \\ &\quad \cdots + \binom{n}{n-1} (1-t) t^{n-1} \mathbf{P}_{n-1} + t^n \mathbf{P}_n, \quad t \in [0, 1],\end{aligned}$$

Bezier Spline Curves

- A common use for Bezier curves is in font definition



Contoh Bezier Curve

- Diberikan suatu kurva bezier derajat 1 dengan titik awal (-4,4) dan titik akhir (0,0).
- Buatlah plotting titik pada kurva sebanyak 6 titik (termasuk titik awal dan akhir).

Degree 1

$$\mathbf{B}(t) = \mathbf{P}_0 + t(\mathbf{P}_1 - \mathbf{P}_0) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1, t \in [0, 1]$$

- $\mathbf{P}_0 = (-4, 4) \quad \mathbf{P}_1 = (0, 0)$
- $k=5; \text{step} = 1/k = 1/5 = 0.2$
- $t=0 \rightarrow (-4, 4) + 0(4, -4) = (4, -4)$
- $t=0.2 \rightarrow (-4, 4) + 0.2(4, -4) = (-3.2, 3.2)$
- $t=0.4 \rightarrow (-4, 4) + 0.4(4, -4) = (-2.4, 2.4)$
- $t=0.6 \rightarrow (-4, 4) + 0.6(4, -4) = (-1.6, 1.6)$
- $t=0.8 \rightarrow (-4, 4) + 0.8(4, -4) = (-0.8, 0.8)$
- $t=1 \rightarrow (-4, 4) + (4, -4) = (0, 0)$

OpenGL Approximation Spline Functions

- Bezier splines and B-splines can be displayed using OpenGL functions
- The core library contains Bezier functions, and GLU has B-spline functions
- Bezier functions are often hardware implemented

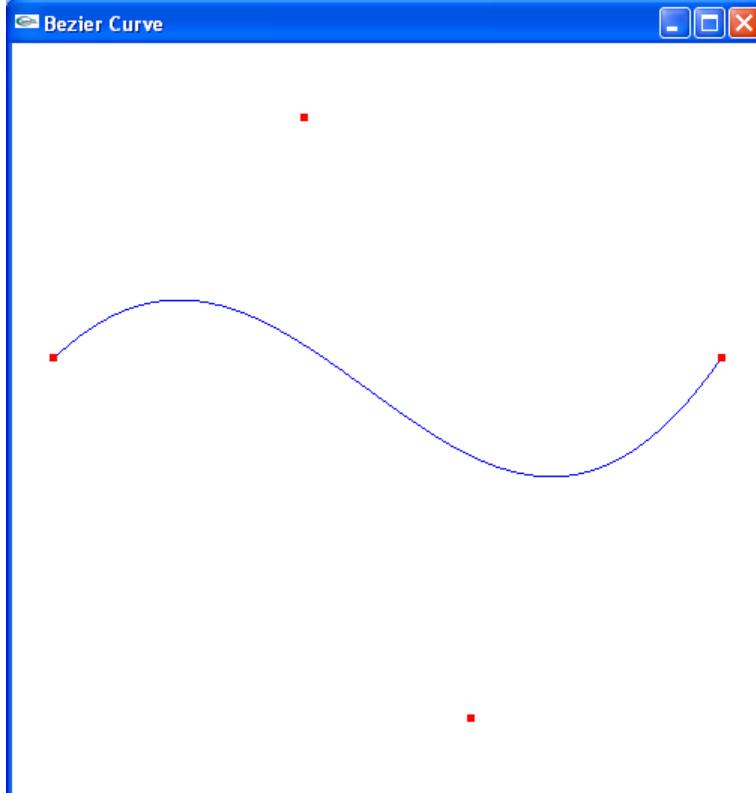
OpenGL Bezier-Spline Curve Functions

- We specify parameters and activate the routines for Bezier-curve display with
 - `glMap1*(GL_MAP1_VERTEX_3, uMin, uMax, stride, nPts, *ctrlPts);`
 - `glEnable(GL_MAP1_VERTEX_3);`
- and deactivate with
 - `glDisable(GL_MAP1_VERTEX_3);`
 - `uMin` and `uMax` are typically 0 and 1.0
 - `stride=3` for 3D
 - `nPts` is the number of control points
 - `ctrlPts` is the array of control points

OpenGL Bezier-Spline Curve Functions

- After setting parameters, we need to evaluate positions along the spline path and display the resulting curve. To calculate coordinate positions we use
`glEvalCoord1*(uValue);`
where uValue is assigned some value in the interval from uMin to uMax

Example OpenGL Code



prog8OGLBezierCurve.cpp

Bezier Surfaces

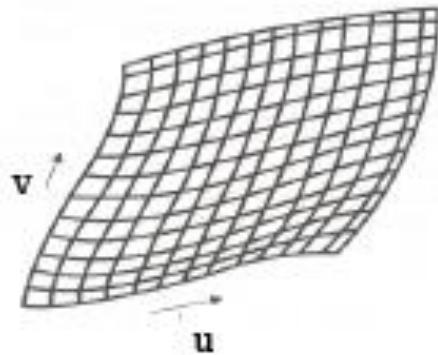
- Two sets of Bezier curves can be used to design an object surface

$$P(u, v) = \sum_{j=0}^m \sum_{k=0}^n p_{j,k} BEZ_{j,m}(v) BEZ_{k,n}(u)$$

with $p_{j,k}$ specifying the location of $(m+1)$ by $(n+1)$ control points

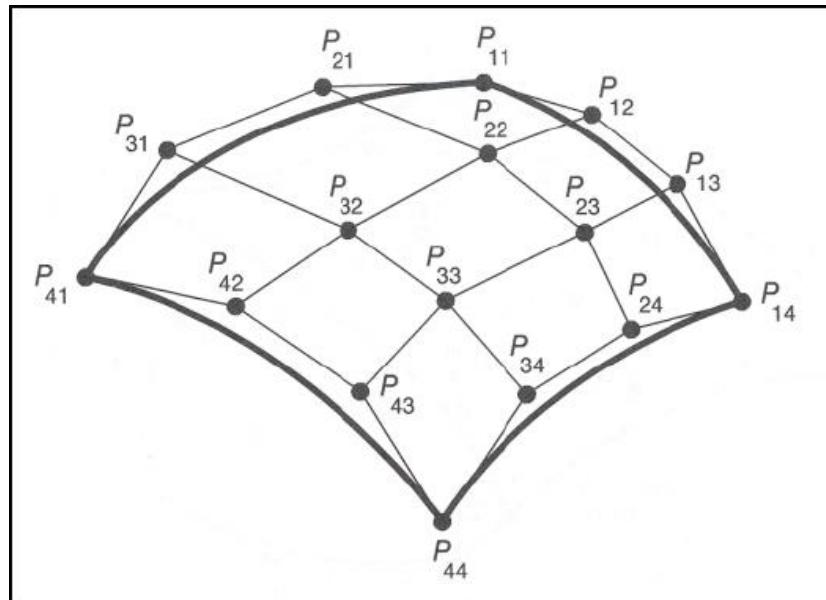
Bezier Surfaces

- u and v parameters



Bezier Surfaces

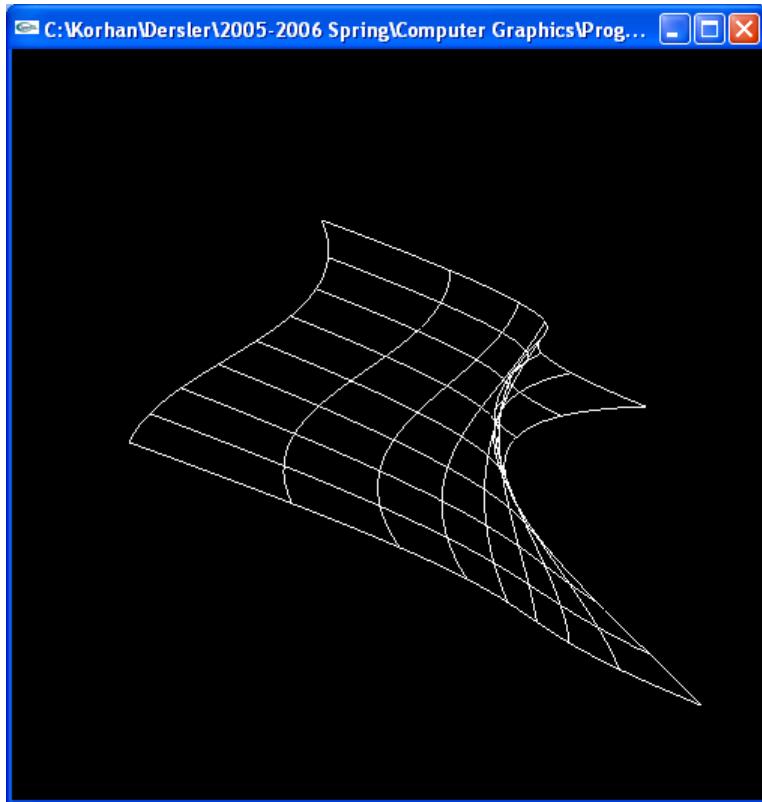
- An example Bezier surface



OpenGL Bezier-Spline Surface Functions

- We specify parameters and activate the routines for Bezier surface display with
 - `glMap2*(GL_MAP2_VERTEX_3, uMin, uMax, uStride,
nuPts, vMin, vMax, vStride, nvPts, *ctrlPts);`
 - `glEnable(GL_MAP2_VERTEX_3);`
- and deactivate with
 - `glDisable(GL_MAP2_VERTEX_3);`
 - `uMin, uMax, vMin and vMax` are typically 0 and 1.0
 - `stride=3` for 3D

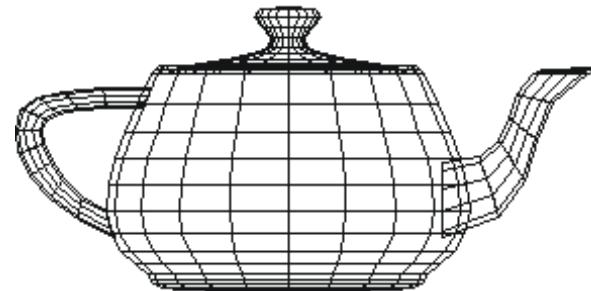
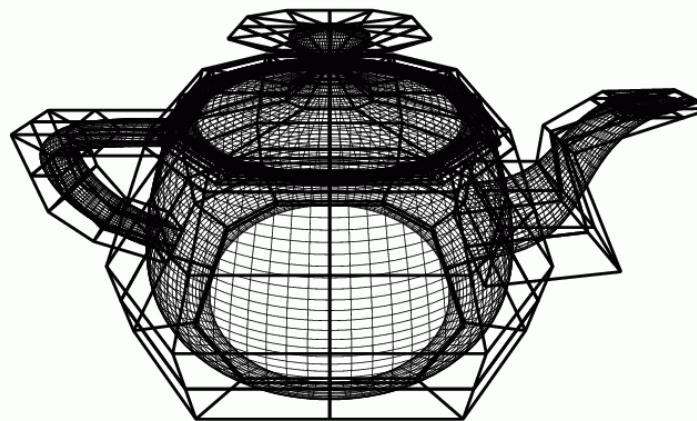
Example OpenGL Code



bezsurf.c

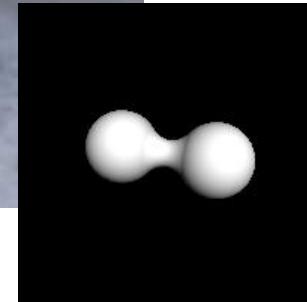
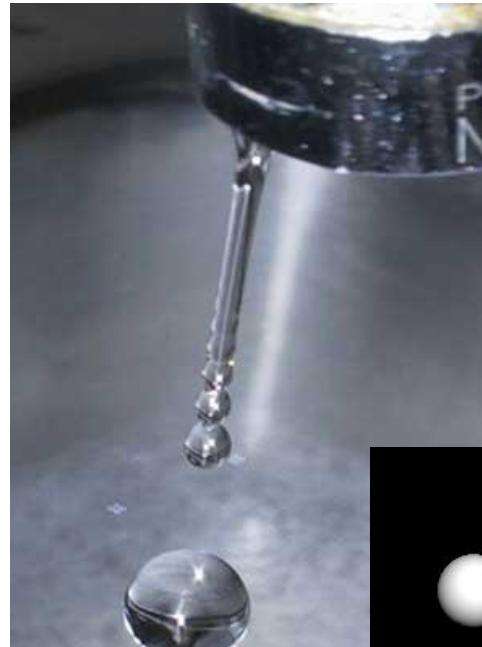
Bézier Surfaces: Example

- Utah Teapot
modeled by 32
Bézier Patches

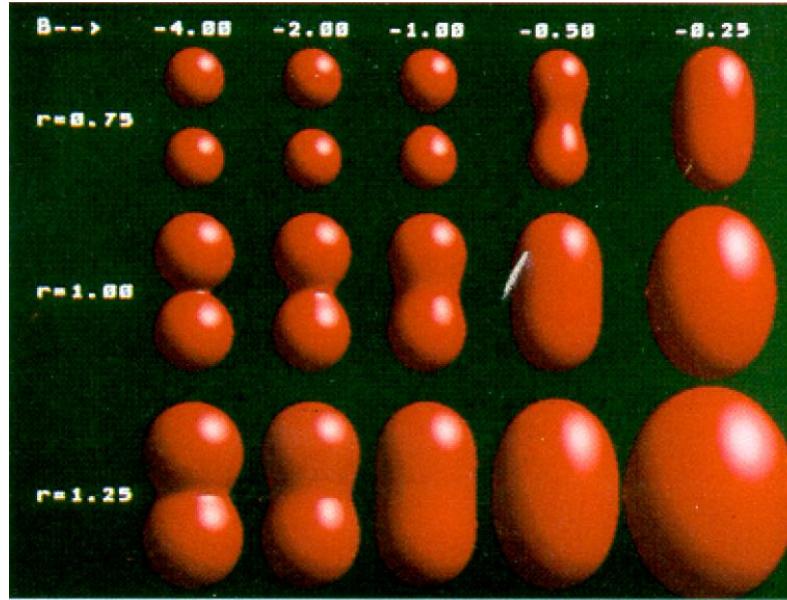


Blobby Objects

- Memodelkan objek yang dapat berubah bentuk tapi volumenya tetap
- Contoh
 - Water drops
 - Molecules
 - Force fields

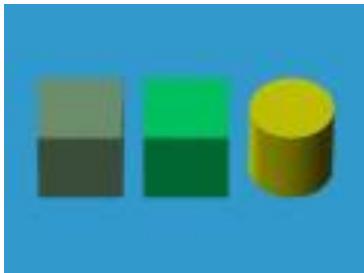


Metaballs (Blinn Blobbies)

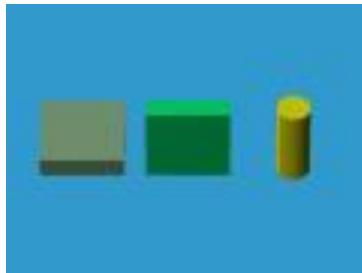


- Blobby objek dengan beberapa nilai radius r dan jarak B
- 2 obyek blobby sifatnya seperti molekul, semakin dekat jaraknya maka akan menyatu

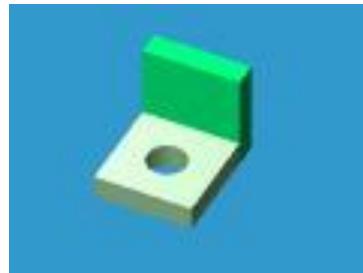
Constructive Solid Geometry



Primitives



Transformed

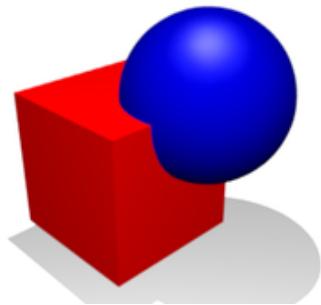


Combined

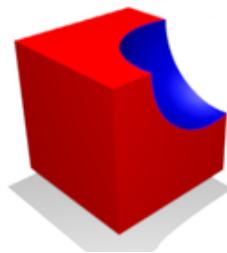
Bermula dari objek geometri primitive, ditransformasikan dan dikombinasikan membentuk objek yang kompleks

Operations in constructive solid geometry

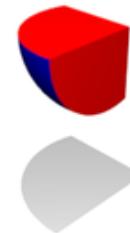
Boolean union



Boolean difference

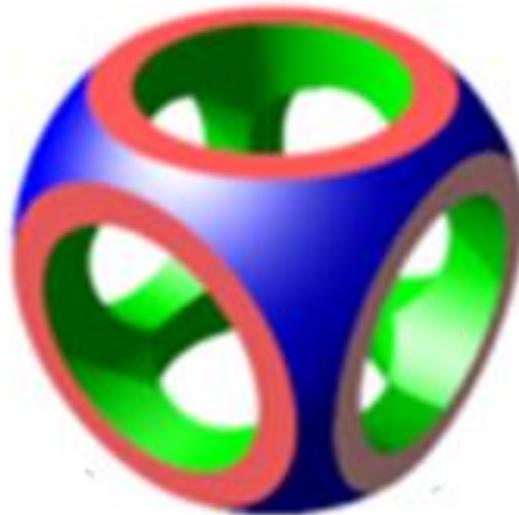


Boolean intersection

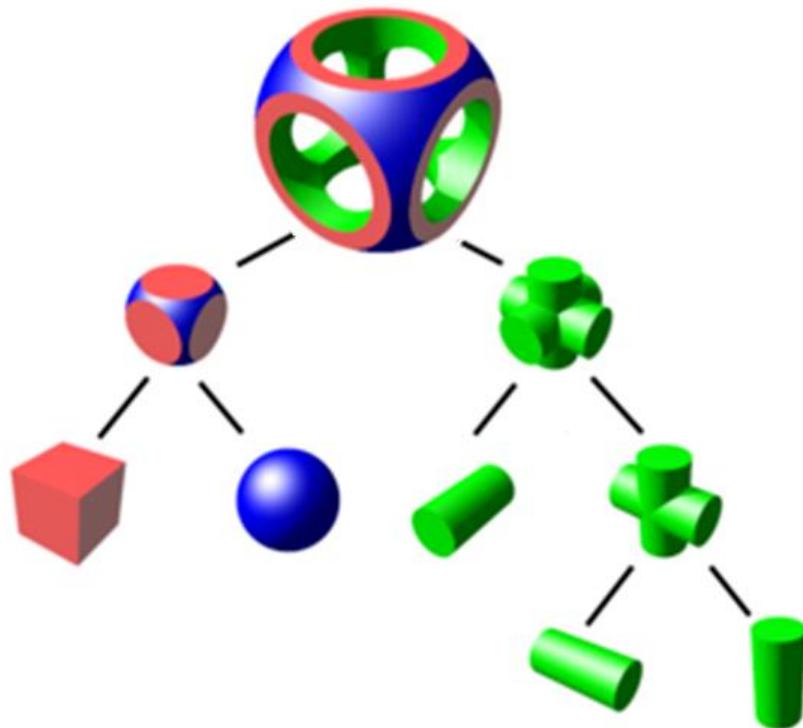


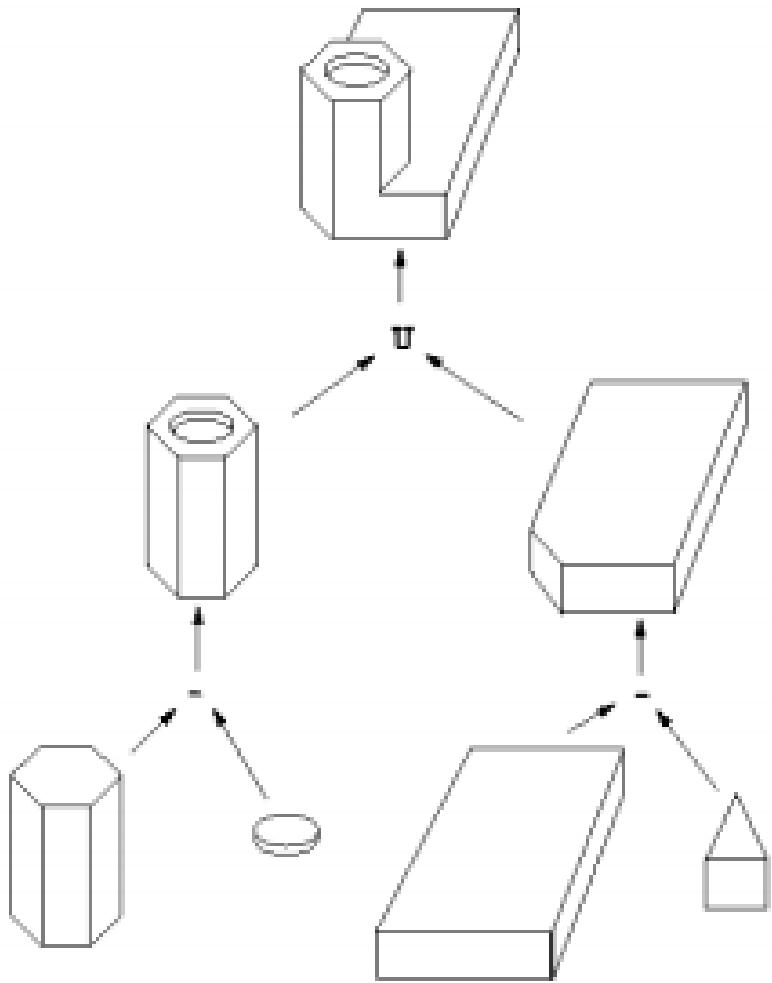
CGS

-



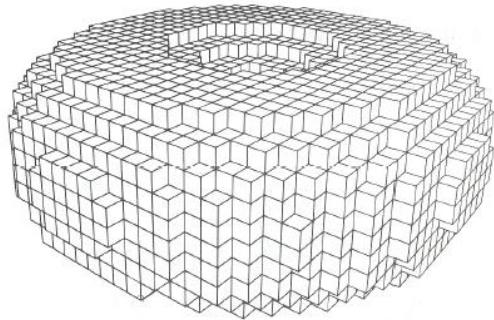
: objek berikut?





Voxel

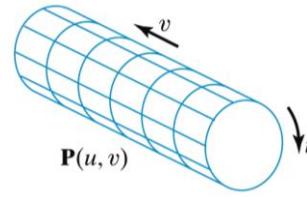
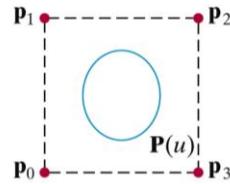
- Membagi ruang menjadi sekumpulan grid berukuran sama
 - Setiap sel pada grid disebut *voxel*
(ingat istilah *pixel*)



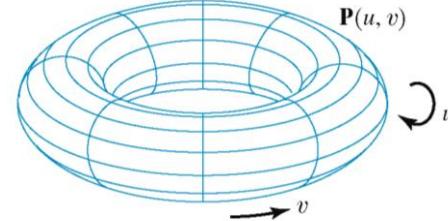
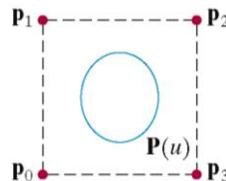
FvDFH Figure 12.20

Sweep (Extrusion)

- Membuat objek 3D dari bidang 2D yang di-sweep pada sumbu atau lintasan tertentu
- Dapat juga membentuk bidang 2D dari suatu kurva

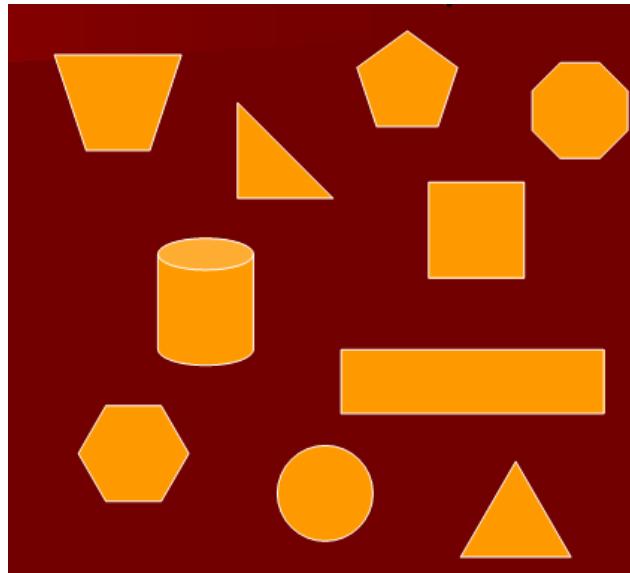


Axis of
Rotation



Geometri

- Trapezoid
- Segitiga
- Persegi
- Persegi panjang
- Segi lima
- Segi enam
- lingkaran
- Silinder



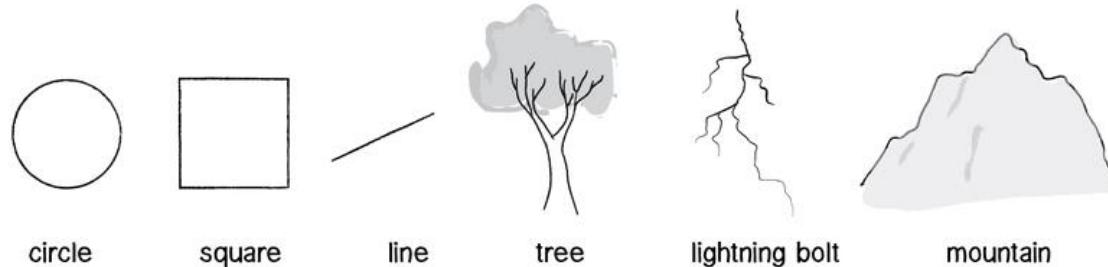
Menjadi pertanyaan

- Dapatkah kita memodelkan objek2 pada alam nyata ini hanya menggunakan geometri2 euclidian?



Geometri

- Secara intuitif geometri euclidian hanya tepat untuk memodelkan objek-objek artifisial contoh: gedung, rumah, monitor, drum
- Objek di alam umumnya memiliki bentuk irregular, pola geometrinya tidak seragam kadang bahkan ada yang tidak beraturan(bersifat infinit details dan infinit length)
- Untuk kebanyakan objek alam geometri euclidian kurang cocok

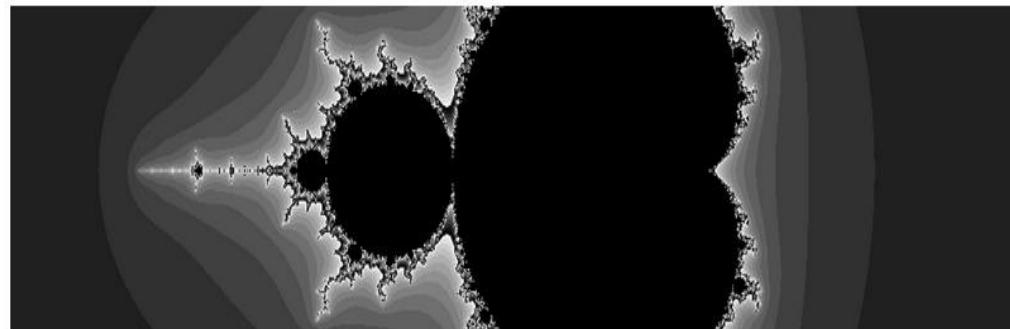


The Fractal Geometry of Nature

The term *fractal* (from the Latin *fractus*, meaning “broken”) was coined by the mathematician Benoit Mandelbrot in 1975. In his seminal work “The Fractal Geometry of Nature,” he defines a fractal as “a rough or fragmented geometric shape that can be split into parts, each of which is (at least approximately) a reduced-size copy of the whole.”

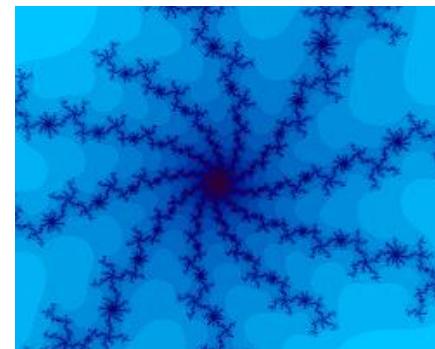
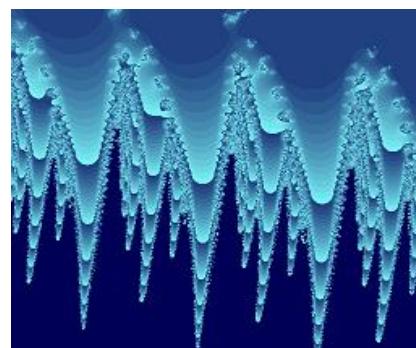
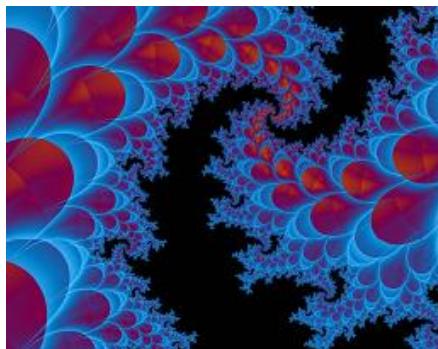
One of the most well-known and recognizable fractal patterns is named for Benoit Mandelbrot himself. Generating the Mandelbrot set involves testing the properties of complex numbers after they are passed through an iterative function. Do they tend to infinity? Do they stay bounded?

---Benoit Mandelbrot (1984)



Fractal

- Merupakan objek yang terbentuk dari pengulangan pola geometri tertentu
- Pola diulang sampai level tertentu maupun tak hingga

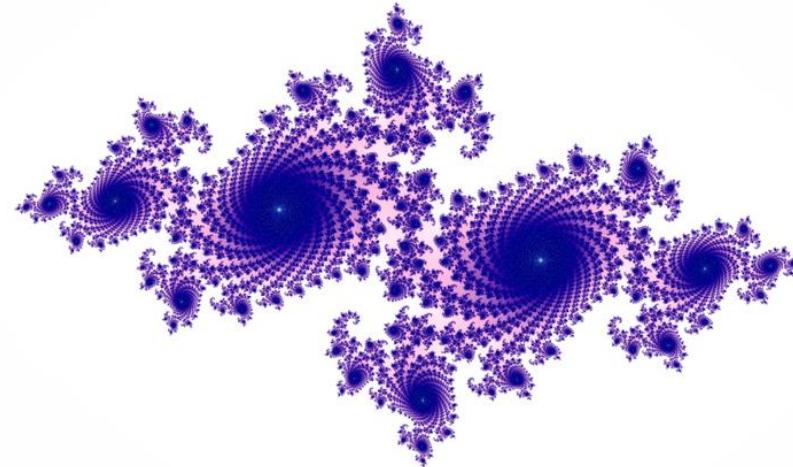


History of Fractal

The first fractals were discovered by a french Mathematician named Gaston Julia who discovered in the early 20th century

Julia's work was rediscovered by Benoit Mandelbrot.

The most famous of all fractals is the **Mandelbrot set**.



Julia set

The Pattern of Math

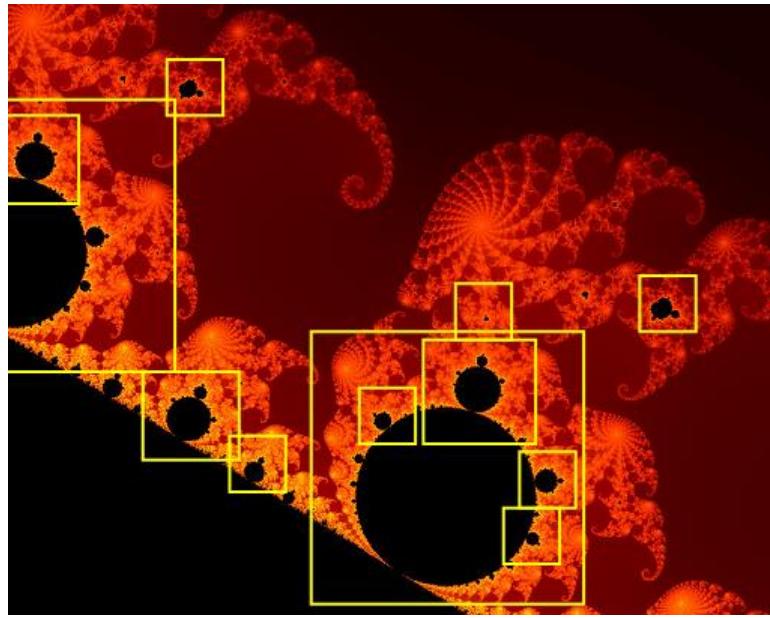
- Mandelbrot set is the set of values of c in the complex plane for which the orbit of 0 under iteration of the complex quadratic polynomial

$$z_{n+1} = z_n^2 + c$$

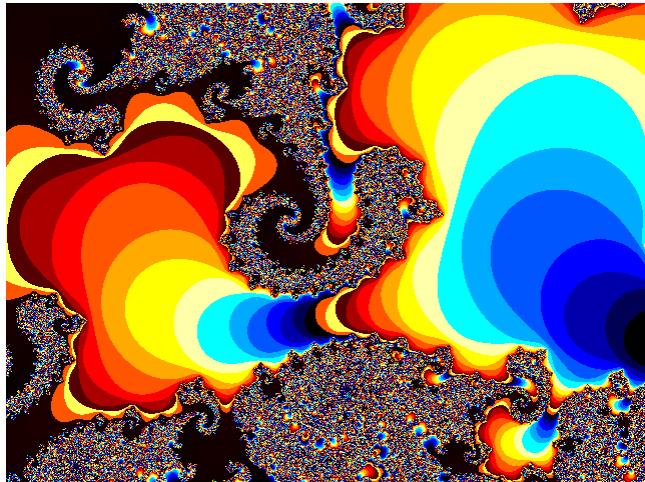
$$c \in M \iff \lim_{n \rightarrow \infty} |z_{n+1}| \leq 2$$



Self similarity

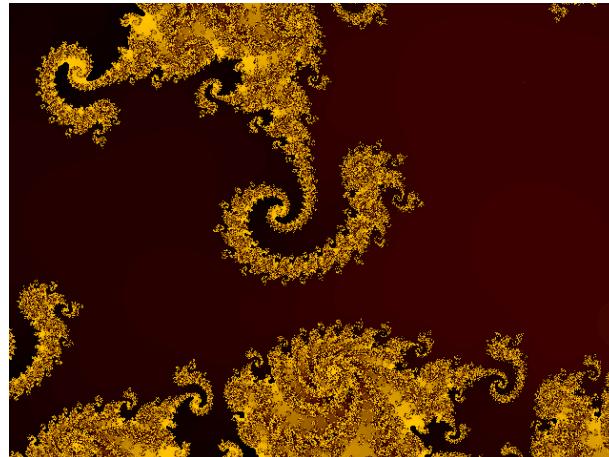


As we magnify the object, we see the same thing over
and over again



These two pictures are interesting because they show the same portion of the mandelbrot set colored differently.

- The choice of color scheme really influences what we see in the picture.
- Is this mathematics or art?



Rekursif

- Let's begin our discussion of recursion by examining the first appearance of fractals in modern mathematics. In 1883, German mathematician George Cantor developed simple rules to generate an infinite set:

1. Start with a line.



2. Erase the middle third of that line

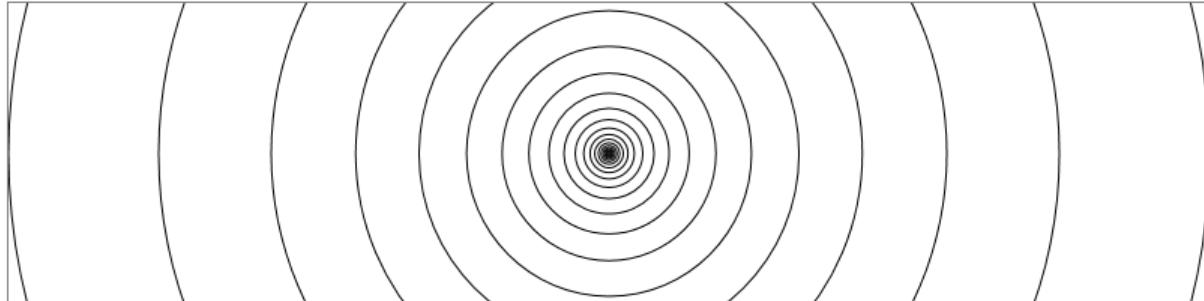


3. Repeat step 2 for the remaining lines,
again and again and again.



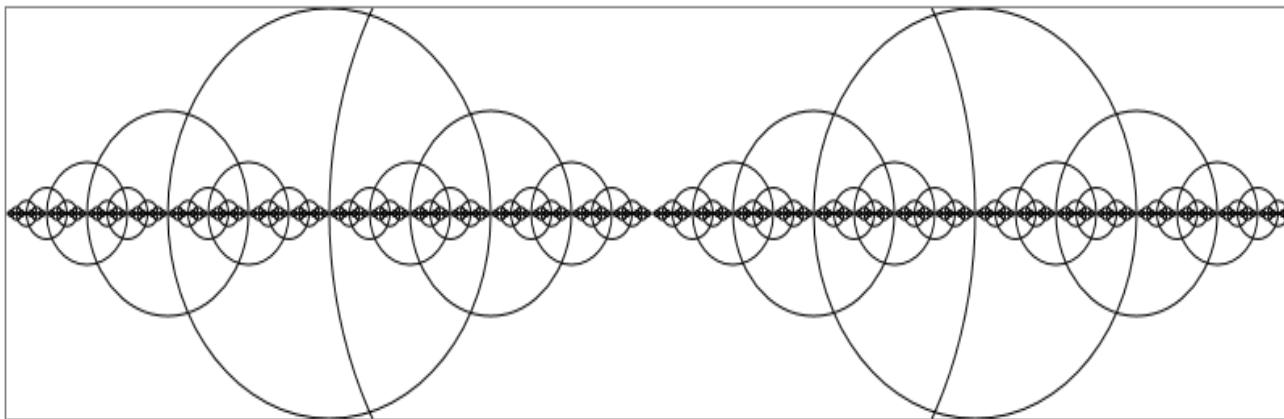
Rekursif

```
void drawCircle(int x, int y, float radius) {  
    ellipse(x, y, radius);  
    if(radius > 0.02) {  
        radius *= 0.75f;  
        drawCircle(x, y, radius); } }
```



Rekursif

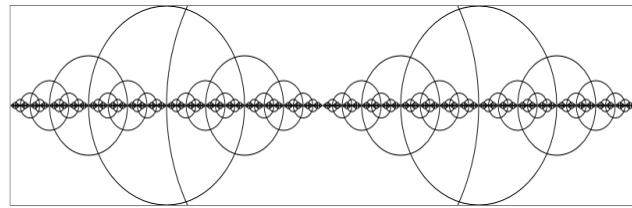
- Bagaimana dengan bentuk berikut:



Rekursif

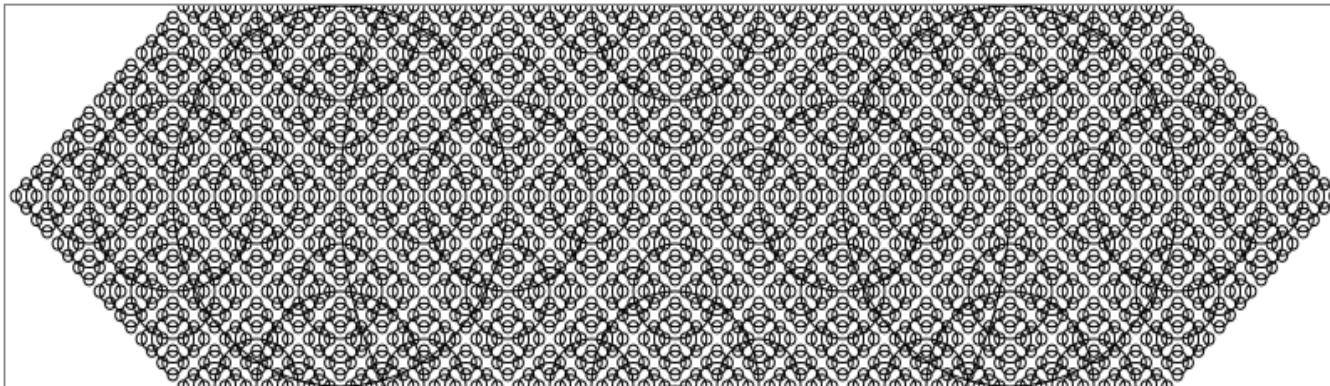
- Bagaimana dengan bentuk berikut:

```
drawCircle(float x, float y, float radius) {  
    ellipse(x, y, radius);  
    if(radius > 0.02) {  
        drawCircle(x + radius, y, radius/2);  
        drawCircle(x - radius, y, radius/2);  
    }  
}
```



Rekursif

- Bagaimana dengan bentuk berikut:



Aristid Lindenmeyer

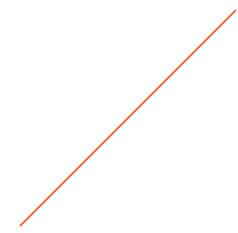
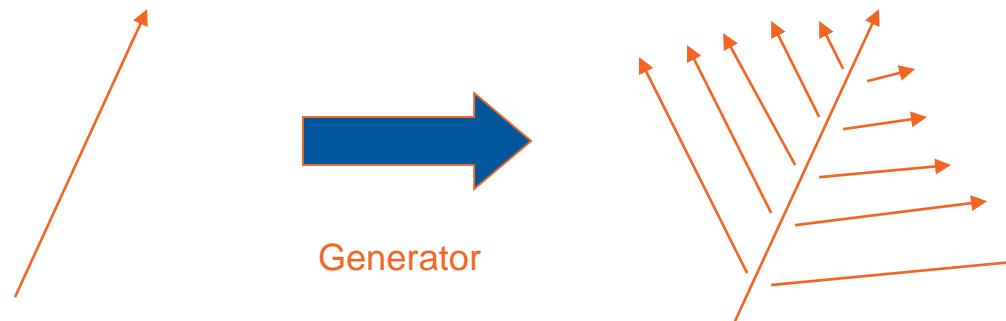
Aristid Lindenmeyer (biologist)
invented L-Systems to model plant
growth



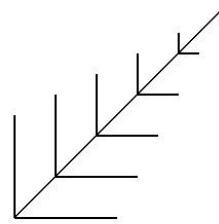
Plant generated using an L-system in 3D



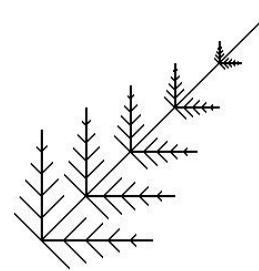
Fractal Fern



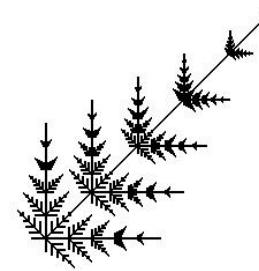
Iteration 0



Iteration 1

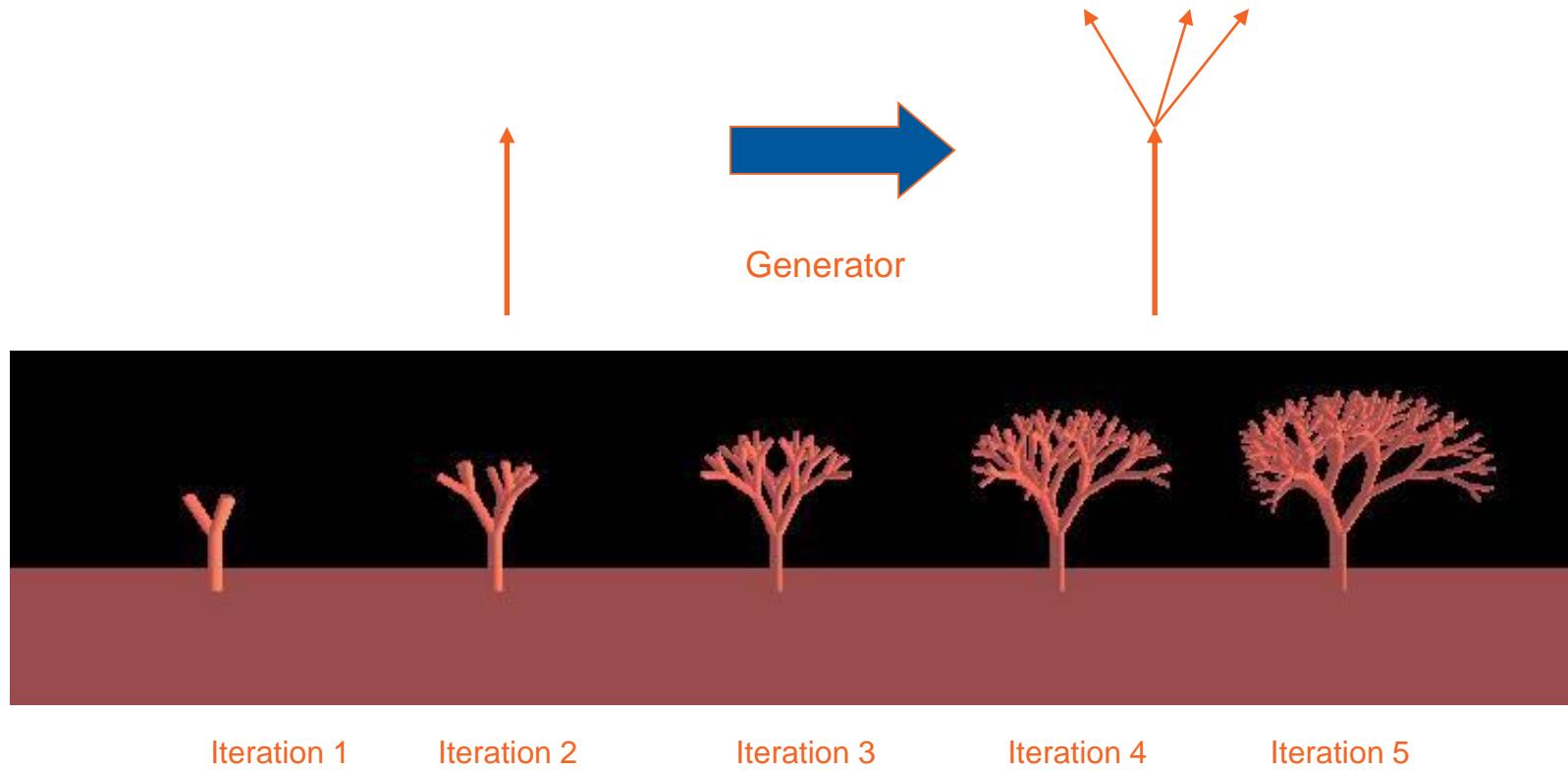


Iteration 2



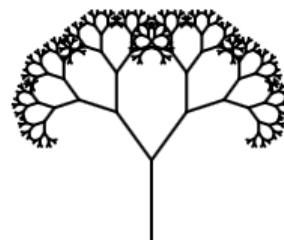
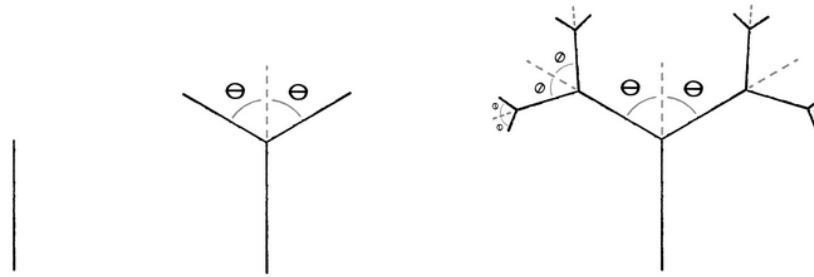
Iteration 3

Fractal Tree



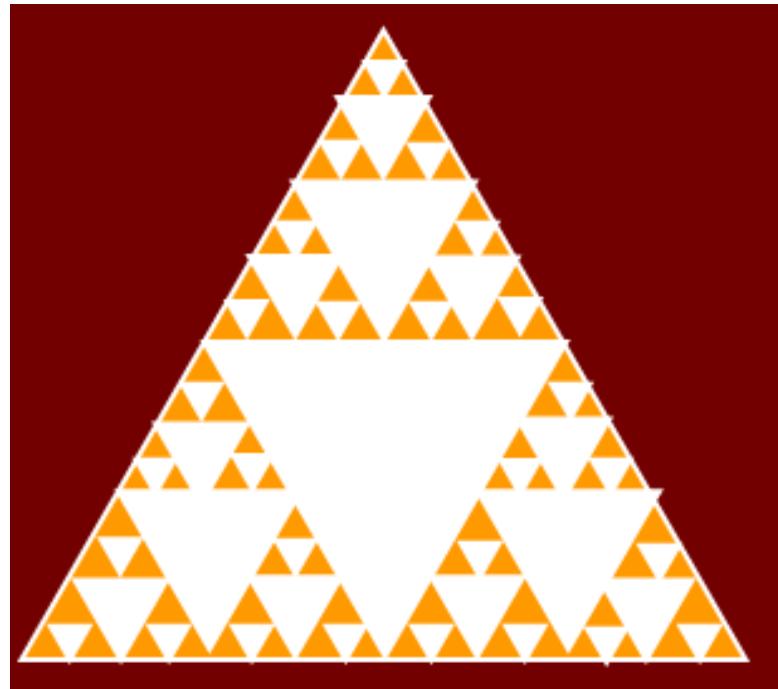
Fractal Tree

1. Draw a line.
2. At the end of the line, (a) rotate to the left and draw a shorter line and (b) rotate to the right and draw a shorter line.
3. Repeat step 2 for the new lines, again and again and again.



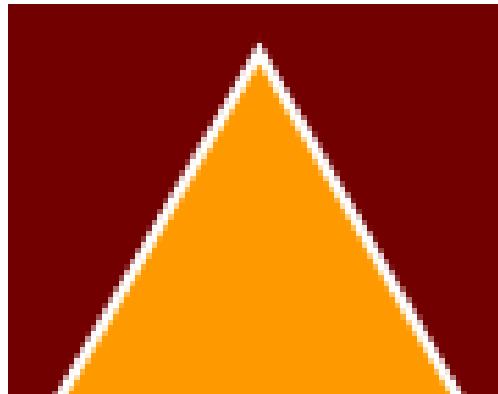
Sierpinski Triangle

A fractal decorative pattern developed by mathematician Wacław Sierpiński

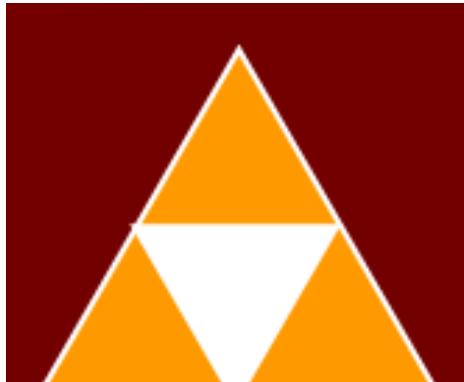


Segitiga Sierpinsky

- Iterasi 1
Buat segitiga sama sisi

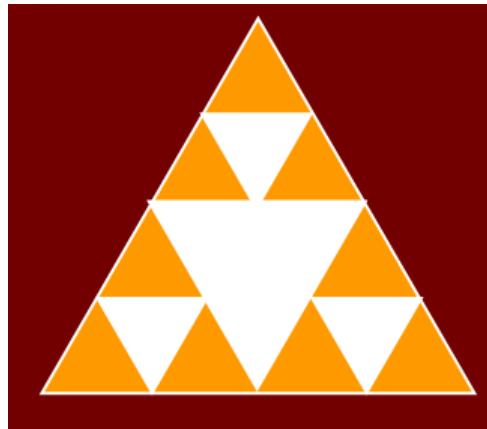


- Iterasi 2 : Buat 3 segitiga yang lebih kecil dengan ukuran lebar dan tinggi setengahnya. Kemudian susun dengan mempertemukan ujung-ujung segitiga. (Putih=area kosong)

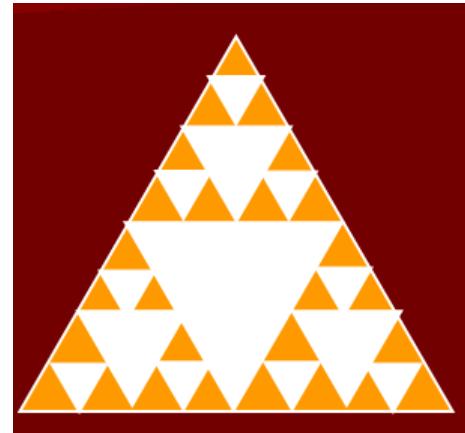


Iterasi selanjutnya

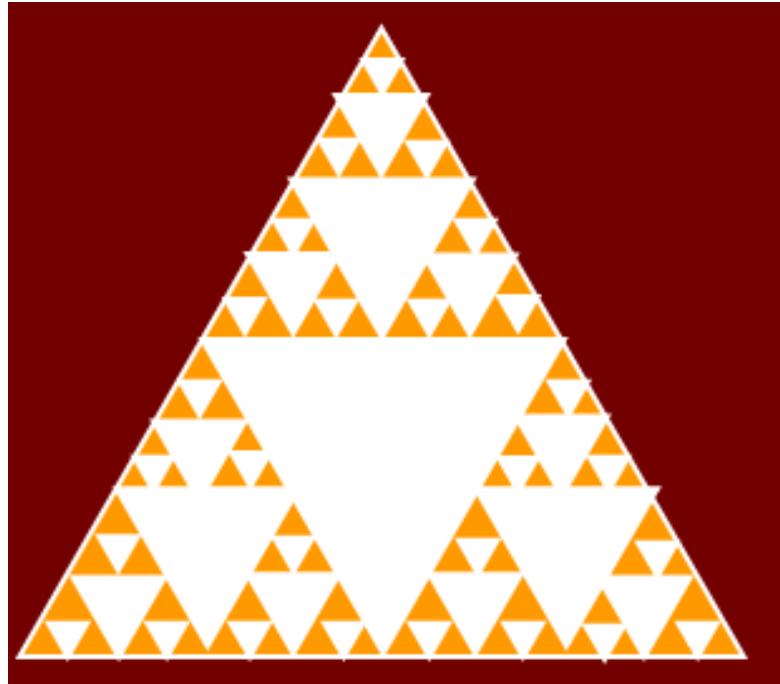
- Iterasi 3



- Iterasi 4

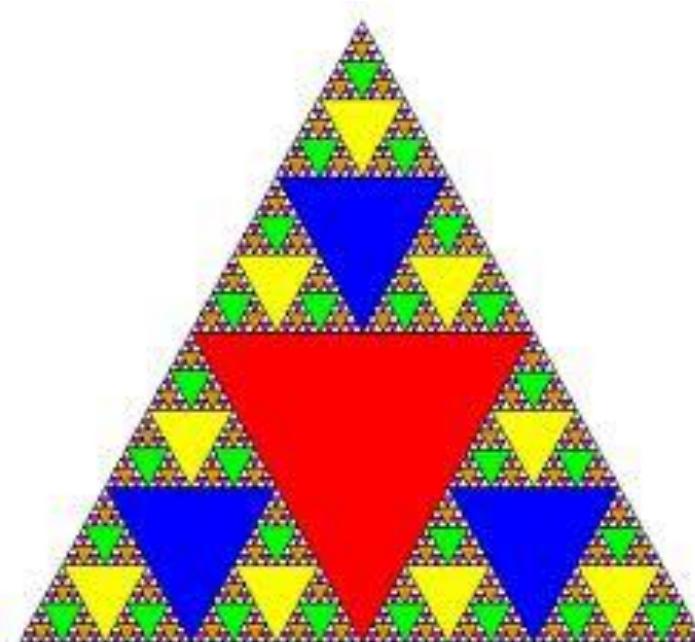


Iterasi ke berapa?



- Iterasi 5

- Sierpinski triangle using different color for each iteration



Segitiga Sierpinski

```
void divide_triangle(point2 a, point2 b, point2 c, int m)
{
    /* triangle subdivision using vertex numbers */
    point2 v0, v1, v2;
    int j;
    if(m>0)
    {
        for(j=0; j<2; j++) v0[j]=(a[j]+b[j])/2;
        for(j=0; j<2; j++) v1[j]=(a[j]+c[j])/2;
        for(j=0; j<2; j++) v2[j]=(b[j]+c[j])/2;
        divide_triangle(a, v0, v1, m-1);
        divide_triangle(c, v1, v2, m-1);
        divide_triangle(b, v2, v0, m-1);
    }
    else(triangle(a,b,c)); /* draw triangle at end of recursion */
}
```

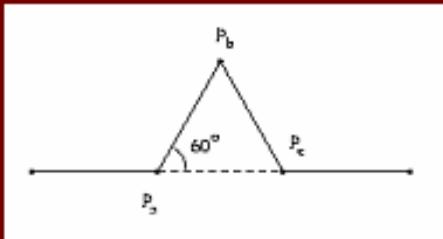
Jika $m=3$, bagaimanakah segitiga sierpinsky yg terbentuk?

Koch Curve



$$\text{Length} = 1$$

First step



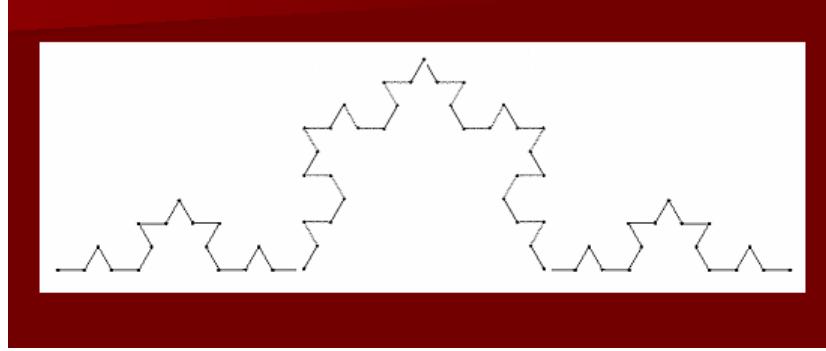
$$\text{Length} = \frac{4}{3}$$

After
2 steps

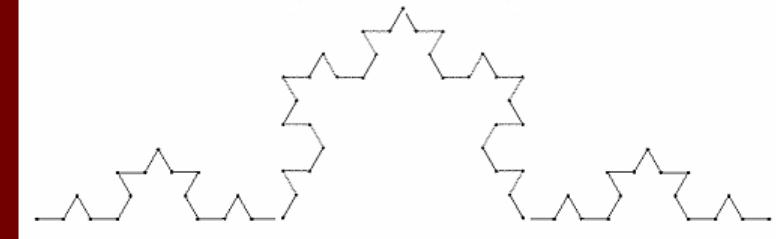


$$\text{Length} = \left(\frac{4}{3}\right)^2$$

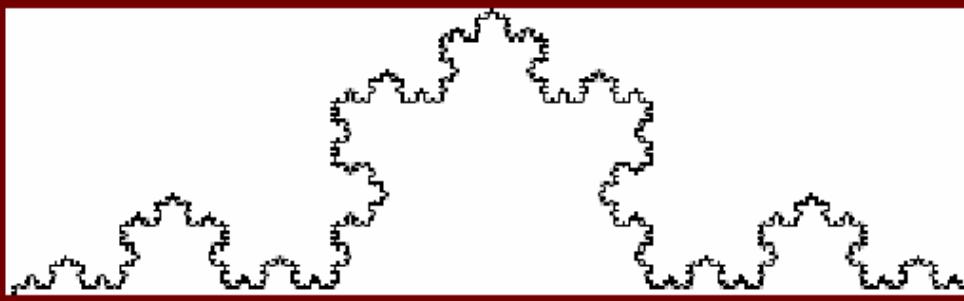
Langkah keberapakah ini?



Berapa panjang sisinya?



$$Length = \left(\frac{4}{3}\right)^3$$



$$\text{Length} = \left(\frac{4}{3} \right)^n$$

Koch Curve

Pseudocode, to draw K_n :

```
If (n equals 0) draw straight line  
else{
```

```
    Draw  $K_{n-1}$ 
```

```
    Turn left 60°
```

```
    Draw  $K_{n-1}$ 
```

```
    Turn right 120°
```

```
    Draw  $K_{n-1}$ 
```

```
    Turn left 60°
```

```
    Draw  $K_{n-1}$ 
```

```
}
```

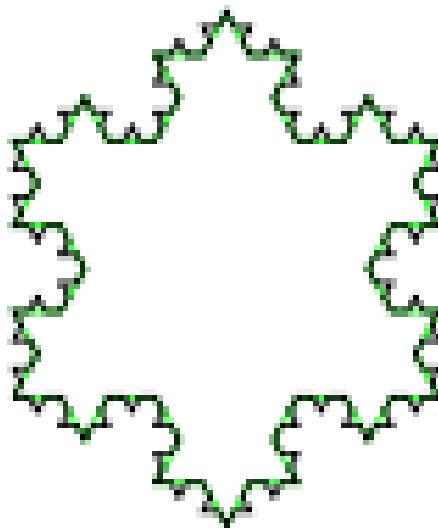
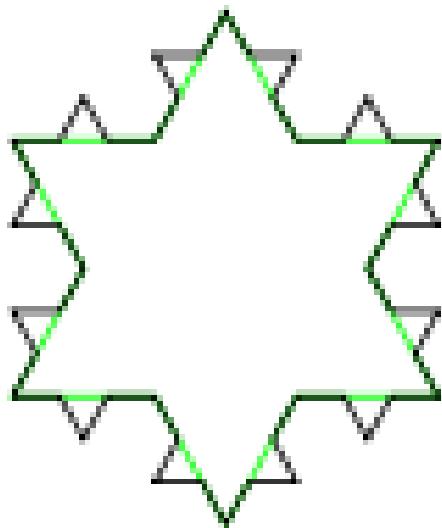
Koch Curve

```
void kochCurve()
{
    glClear(
        GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINES);
    mOldX = 0.0;
    mOldY = 0.0;
    drawKoch(0.0, 1.0, maxLevel );
}
```

```
void drawKoch( GLdouble dir, GLdouble len, GLint n)
{
    double dirRad = 0.0174533 * dir; /* convert to radians */
    float newX = mOldX + len * cos(dirRad);
    float newY = mOldY + len * sin(dirRad);
    if (n == 0)
    {
        glVertex2f(mOldX, mOldY);
        glVertex2f(newX, newY);
        mOldX = newX;
        mOldY = newY;
    } else {
        n--;
        /* reduce the order */
        len /= 3.0; /* reduce the length */
        drawKoch(dir, len, n);
        dir += 60.0;
        drawKoch(dir, len, n);
        dir -= 120.0;
        drawKoch(dir, len, n);
        dir += 60.0;
        drawKoch(dir, len, n);
    }
}
```

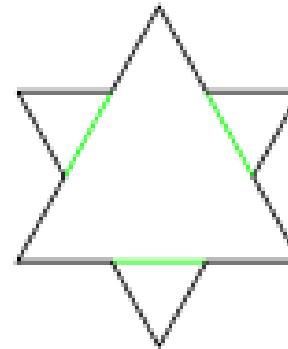
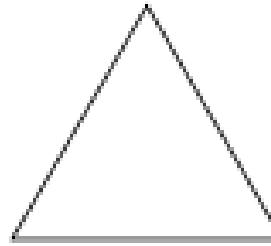
Koch Snowflake

- 3 buah koch curve digabung membentuk segitiga



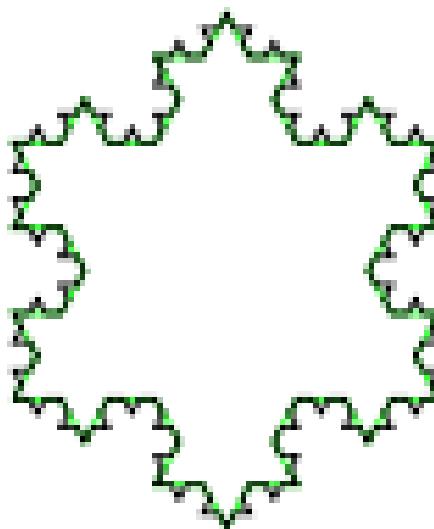
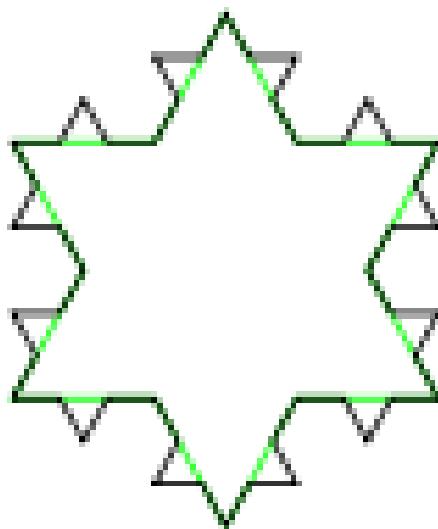
Koch Snowflake

- Buat segitiga
- Bagi tiap sisi menjadi 3, kemudian buat segitiga lagi pada bagian tengah sisi tsb ke arah luar



Koch Snowflake

- Lanjutkan langkah tsb untuk setiap sisi sampai level tertentu.



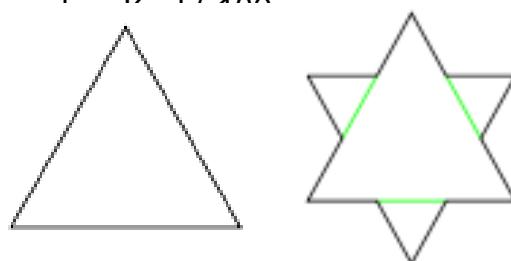
Koch Curve

```
void kochCurve()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINES);

    mOldX = 0.0; mOldY = 0.0;
    drawKoch(60.0, 1.0, maxLevel );

    mOldX = 0.5; mOldY =
        sqrt(3)/2.0;
    drawKoch(-60.0, 1.0, maxLevel);

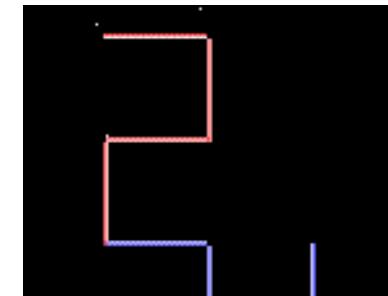
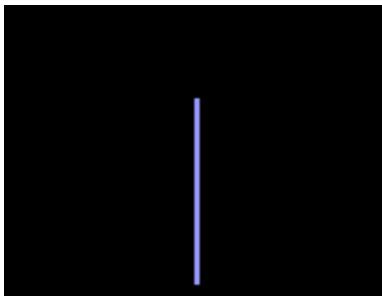
    mOldX = 1.0; mOldY = 0.0;
```

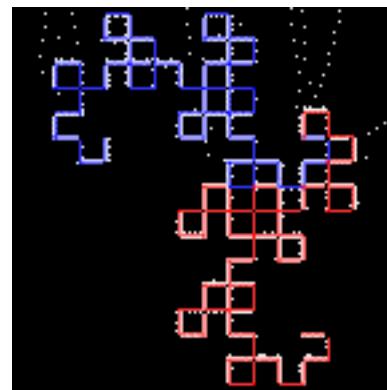
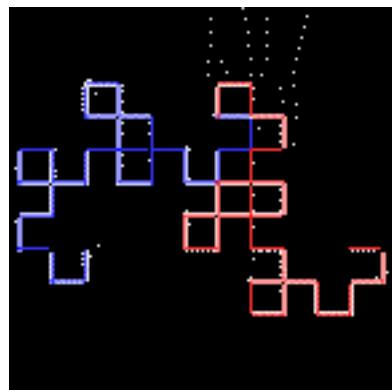
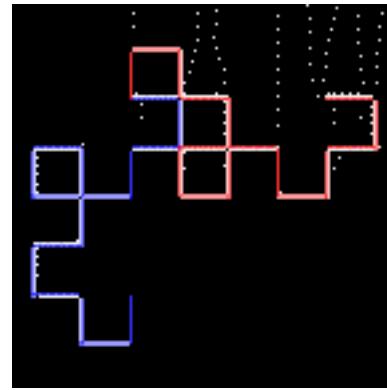
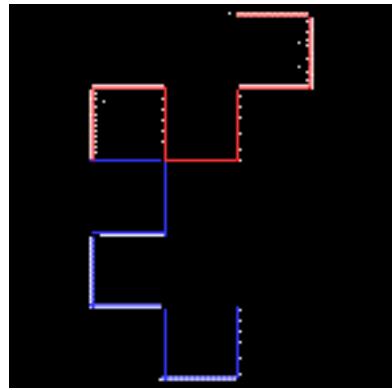


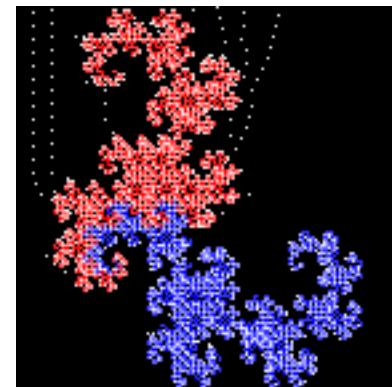
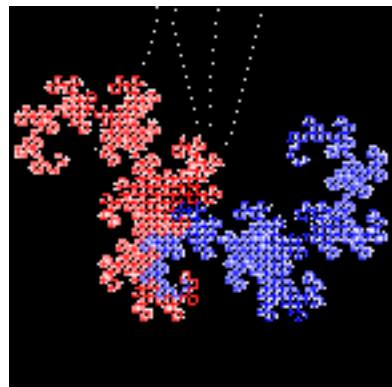
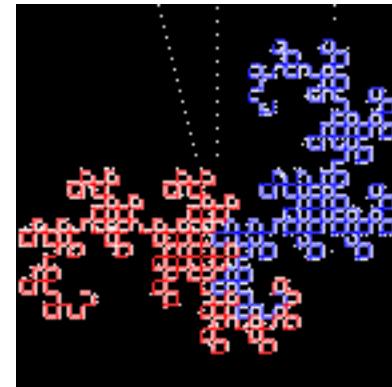
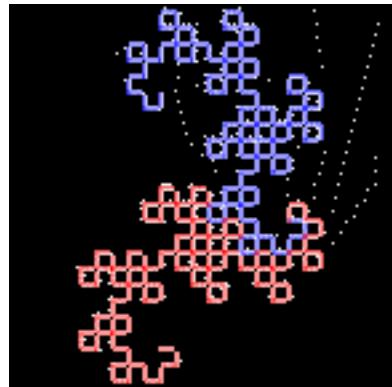
```
void drawKoch( GLdouble dir, GLdouble len, GLint n)
{
    double dirRad = 0.0174533 * dir; /* convert to radians */
    float newX = mOldX + len * cos(dirRad);
    float newY = mOldY + len * sin(dirRad);
    if (n == 0)
    {
        glVertex2f(mOldX, mOldY);
        glVertex2f(newX, newY);
        mOldX = newX;
        mOldY = newY;
    } else {
        n--;
        /* reduce the order */
        len /= 3.0; /* reduce the length */
        drawKoch(dir, len, n);
        dir += 60.0;
        drawKoch(dir, len, n);
        dir -= 120.0;
        drawKoch(dir, len, n);
        dir += 60.0;
        drawKoch(dir, len, n);
    }
}
```

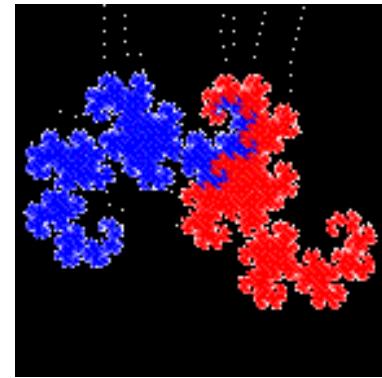
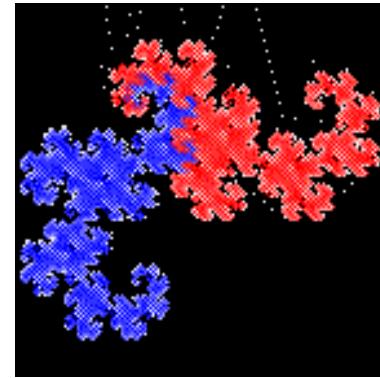
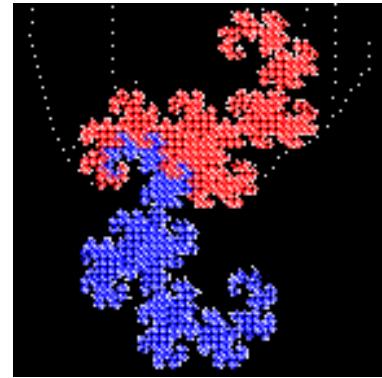
Dragon Curve

- Gambar segmen garis vertikal.
- Tandai titik awal (misalnya, atas) dan titik akhir segmen.
- Putar 90 derajat (berlawanan) dengan titik akhir kemudian gambar bersama gambar asli dan gambar yang sudah diputar.
- Tandai titik akhir dari gambar yang sudah diputar tadi. Putar kembali gambar 90 derajat, dan gabungkan gambar asli dengan dengan gambar yang sudah diputar.









Dragon Curve

Gambar yang asli digambarkan dalam warna biru dan gambar yang sudah diputar merah.

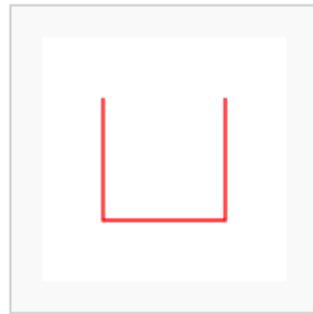
Masing-masing gambar berturut-turut menyusut untuk membuatnya sesuai ukuran yang sama di dalam persegi.

Namun, ketebalan garis selalu sama

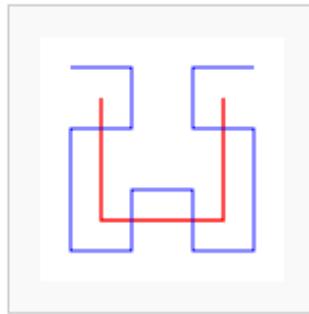
Selain Koch Snowflake, Koch Curve,
Sierpinsky Triangle, dan Dragon Curve,
masih ada lagi jenis-jenis fractal yang lain.

Jenis-jenis fractal tsb dibuat dengan
rumusan tertentu

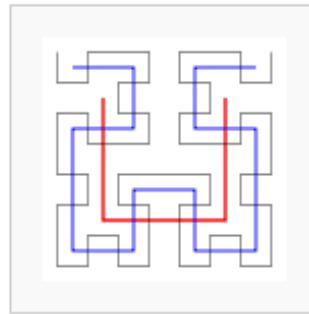
Hilbert Curve



Hilbert curve, first order



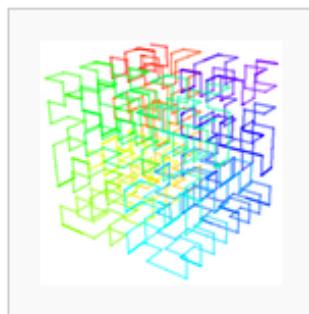
Hilbert curves, first and second orders



Hilbert curves, first to third orders



Hilbert curve in three dimensions



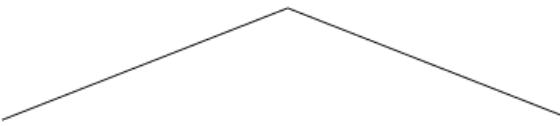
3-D Hilbert curve with color showing progression

Penggunaan Fractal

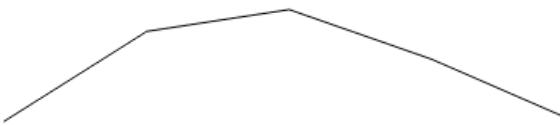


Terrain Fractal

Let's look at an example. Here, we start with a line from -1.0 to 1.0 in X, with the Y value at each endpoint being zero. Initially we'll set the random number range to be from -1.0 to 1.0 (arbitrary). So we generate a random number in that range, and displace the midpoint by that amount. After doing this, we have:



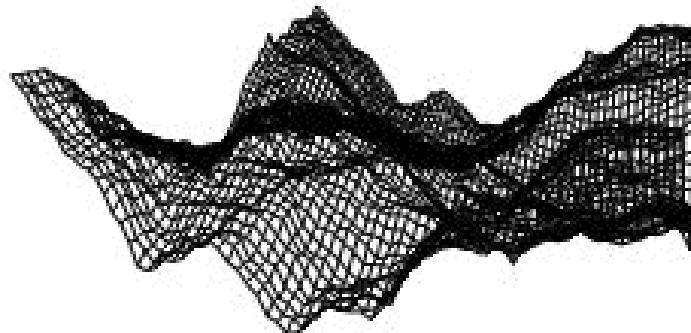
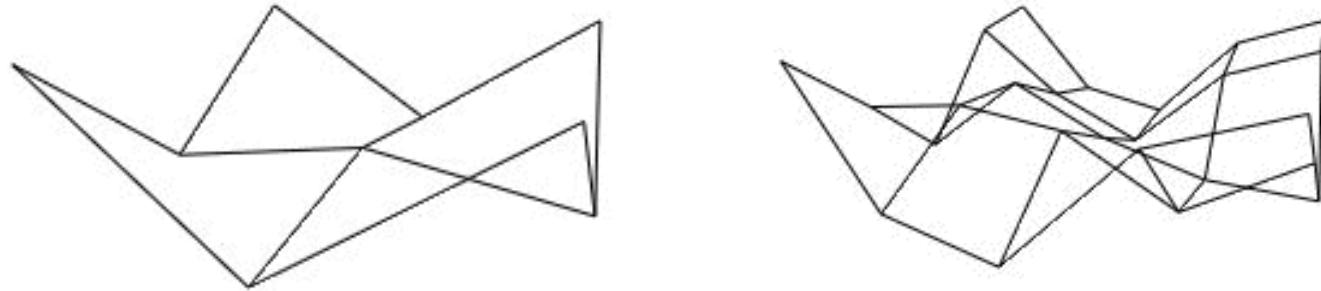
Now the second time through the outer loop, we have two segments, each half the length of the original segment. Our random number range is reduced by half, so it is now -0.5 to 0.5. We generate a random number in this range for each of the two midpoints. Here's the result:



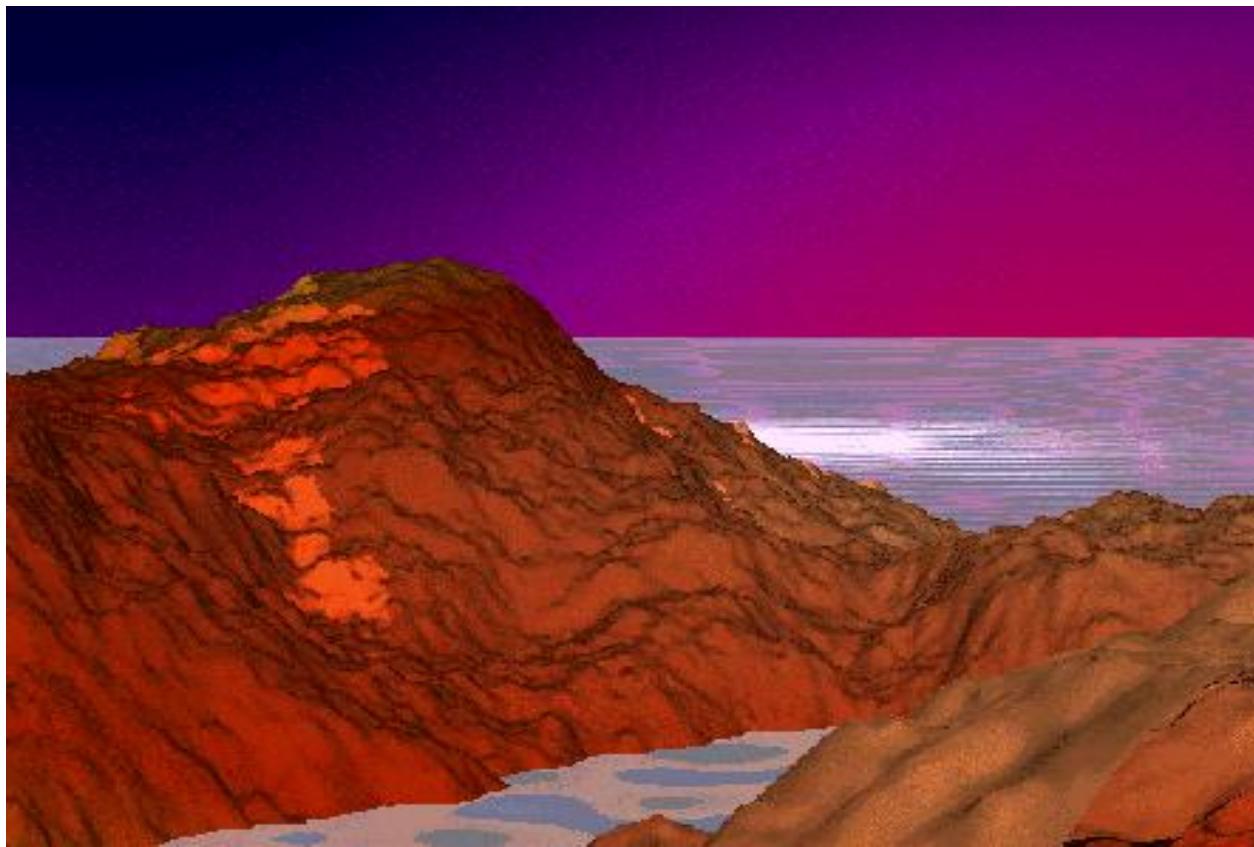
We shrink the range again; it is now -0.25 to 0.25. After displacing the four midpoints with random numbers in this range, we have:



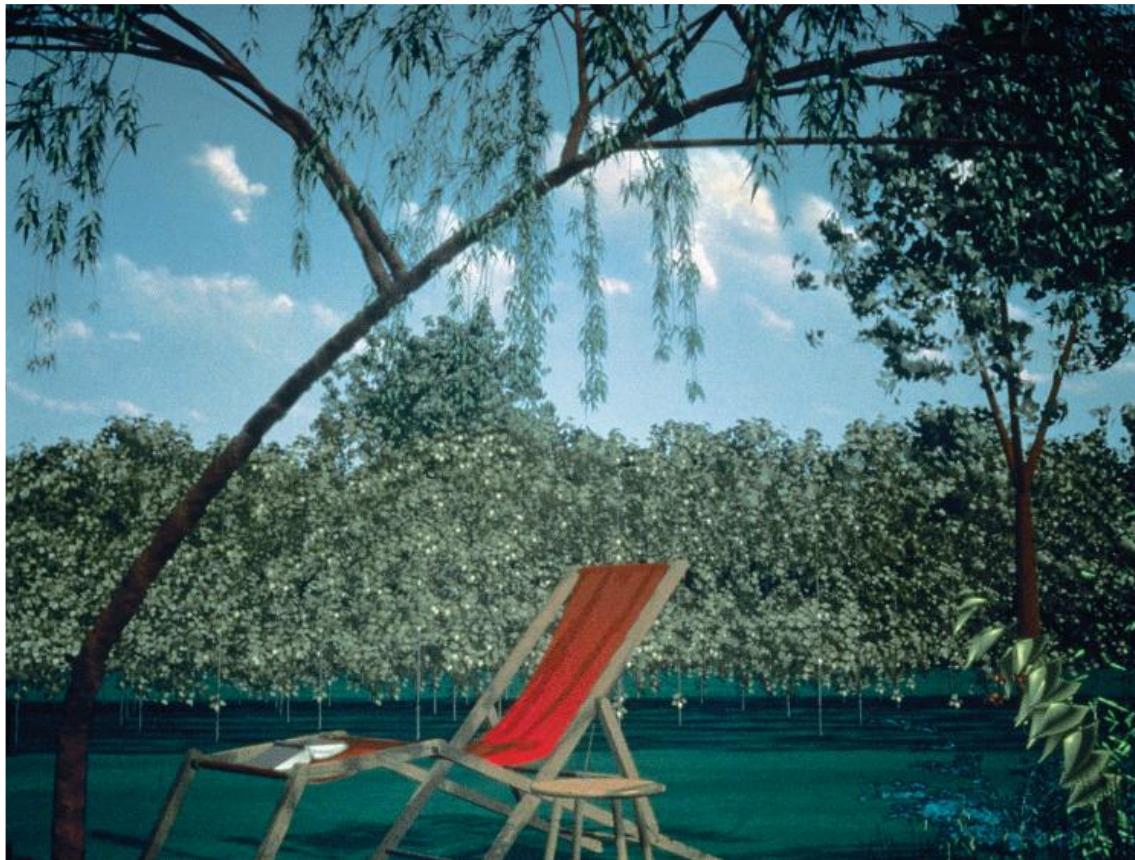


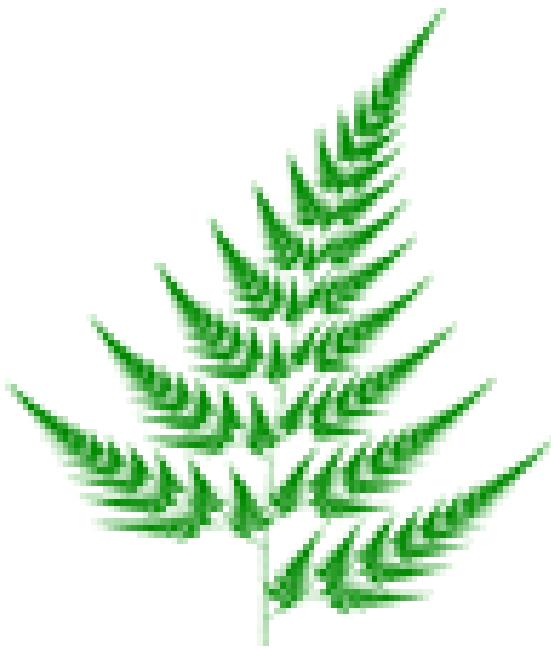


2/15/2006

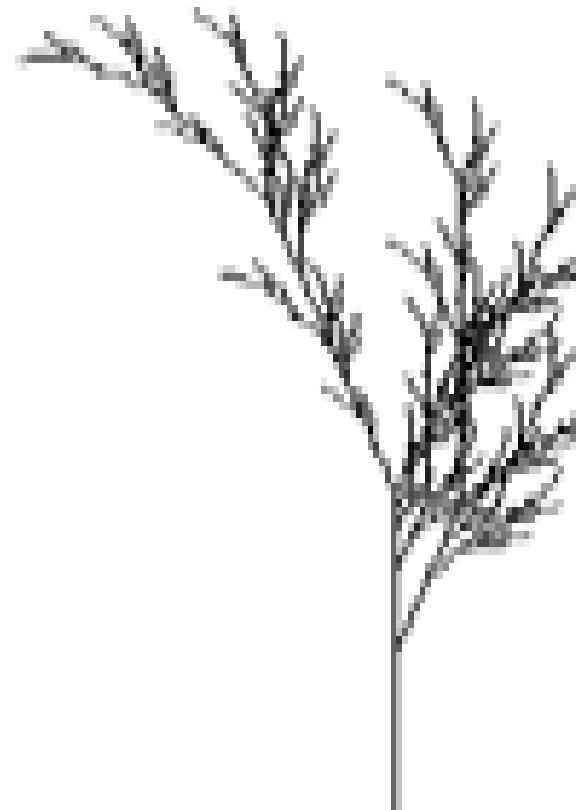


- Mengambar pepohonan



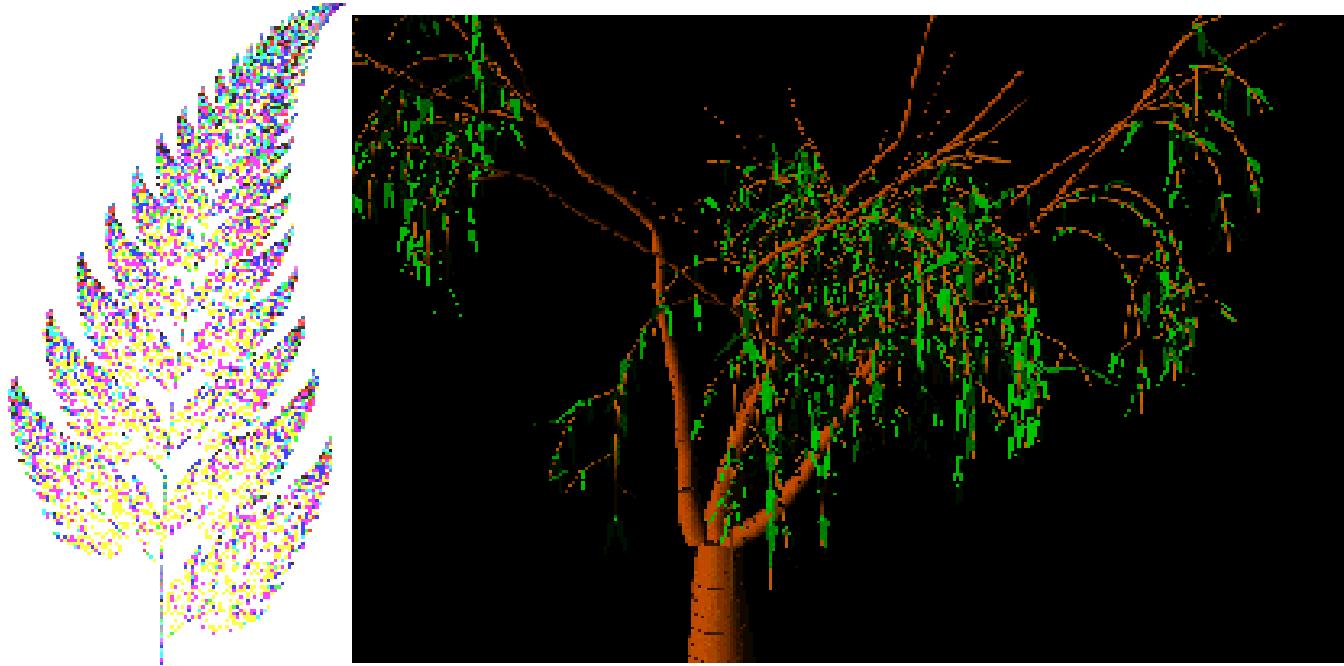


118



TCSS458A Isabelle
Bichindaritz

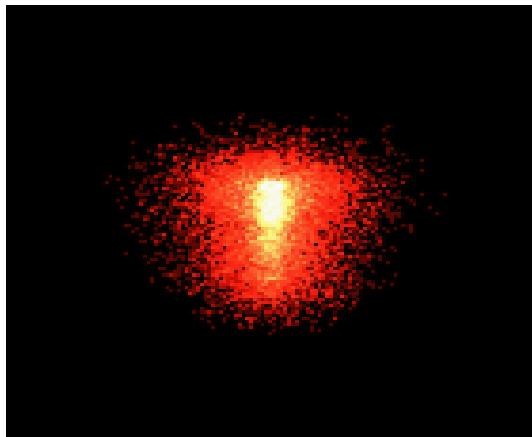
OTHER MODIFICATIONS PRODUCE



Particle System

Particle Systems

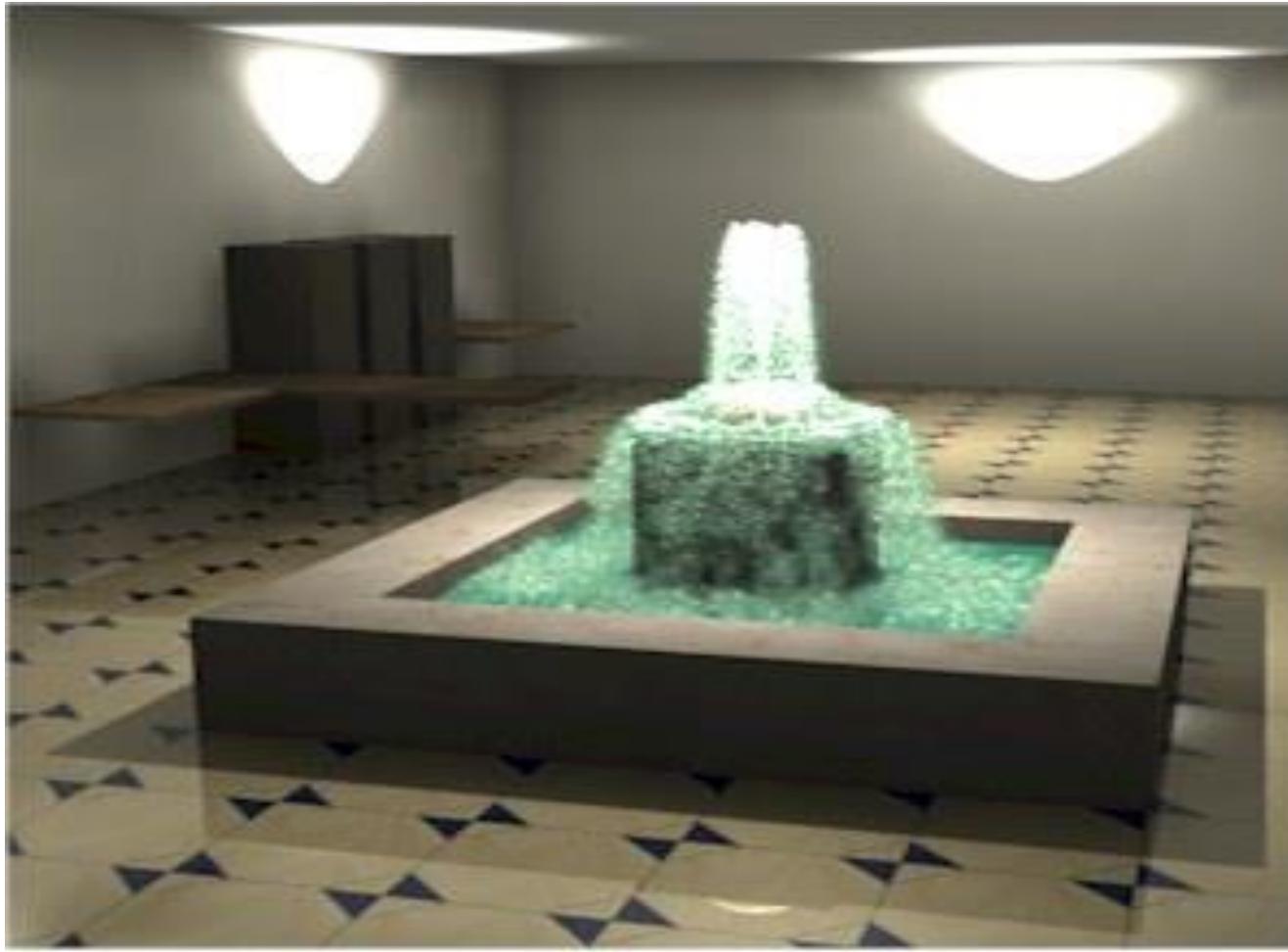
- Jika fractal mememodelkan bentuk yang relatif teratur, bagaimana jika ingin menggambar obyek yang tidak teratur bentuknya? Contoh : kembang api, asap, air
- Pada particle system, objek tidak teratur, tapi ada suatu aturan yang mempengaruhi gerak partikelnya



Definition

- Particle system adalah kumpulan dari sejumlah individual elements atau *particles*.
- Particle systems mengontrol sejumlah particle yang independen(bergerak sendiri-sendiri) tetapi memiliki sifat yang sama

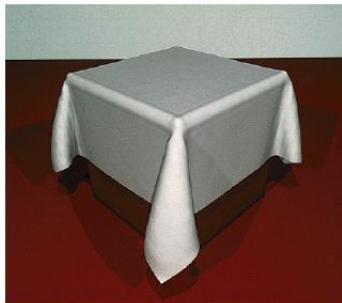
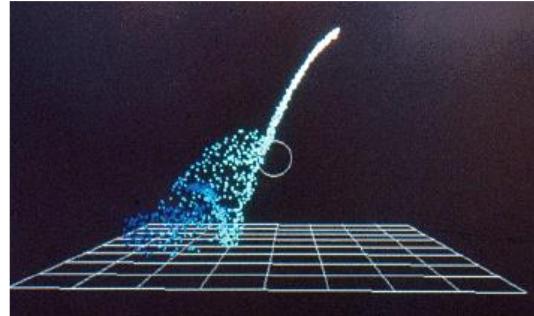
2/15/2006



Bichindaritz

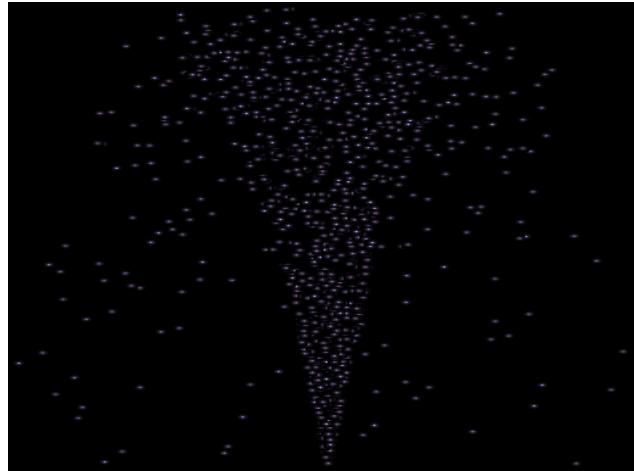
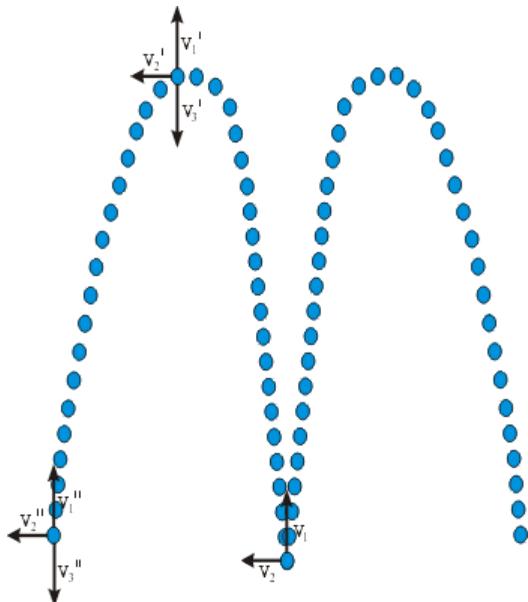
Phisically Based Modelling

- Memodelkan bentuk berupa partikel-partikel yang memiliki sifat fisika



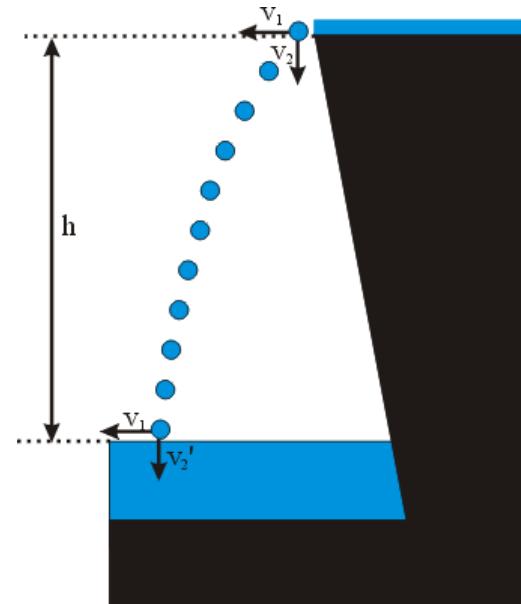
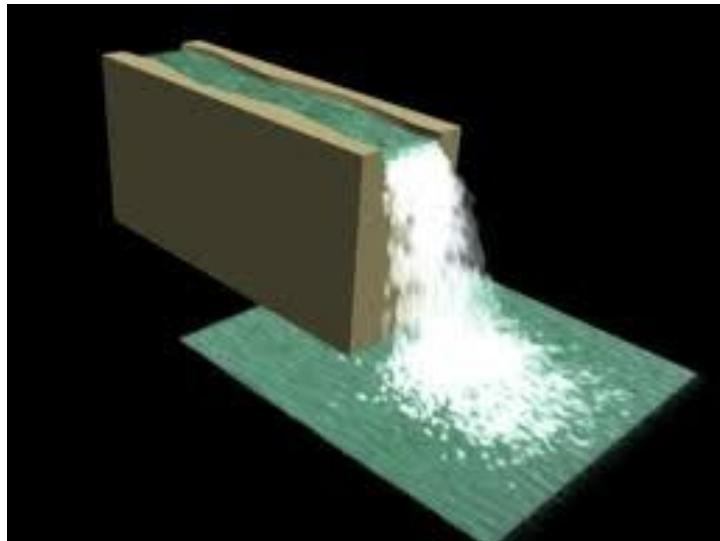
Gerak Partikel

- Air Mancur : Menggambar objek partikel sebanyak n partikel. Masing-masing partikel bergerak sebagaimana gerak air mancur

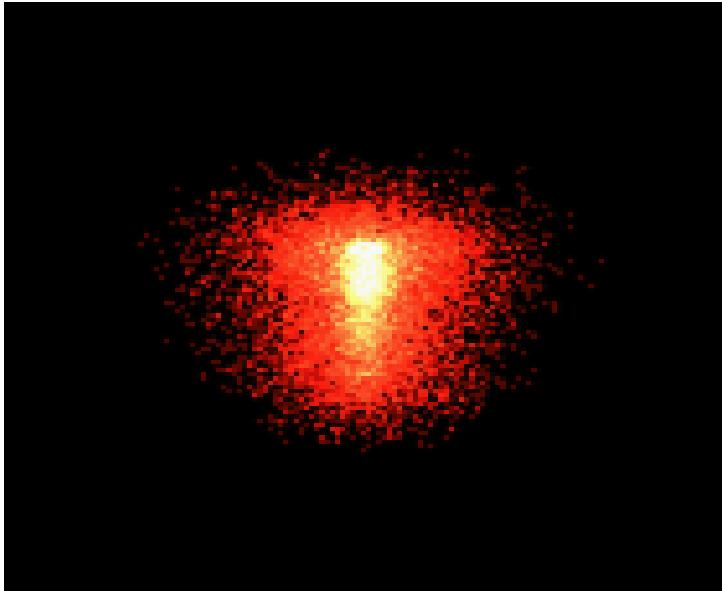


Gerak Partikel

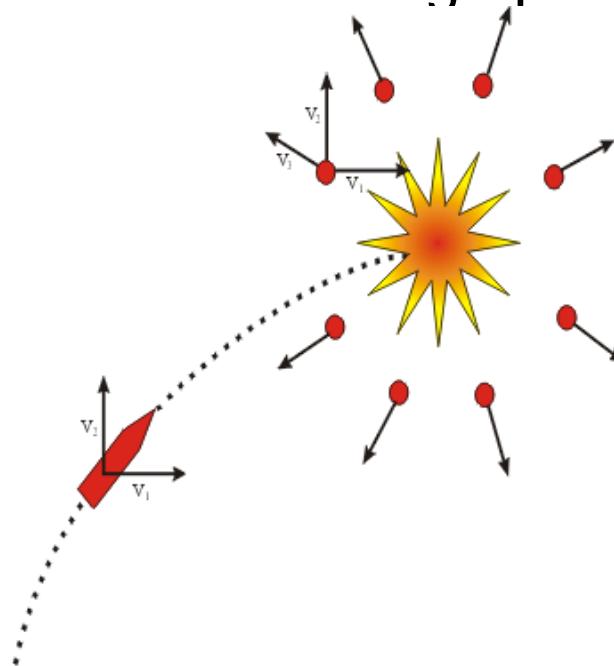
- Air Terjun : Sekian ribu partikel air bergerak sebagaimana rumusan fisika air terjun



Gerak Partikel



- Gerak partikel kembang api



Basic Particle System Physics

- Particle is a point in 3D space.
- Forces (e.g. gravity or wind) accelerate a particle.
- Acceleration changes velocity.
- Velocity changes position

Particle Attributes

Contoh atribut yang dimiliki particle

- Position
- Velocity
- Life Span
- Size
- Weight
- Representation
- Color
- Owner

How to represent particles?

- Points
- Lines
- Texture-mapped quads

Bagaimana Programnya

- Membuat particle sebanyak max particle
- Menggerakkan partikel sesuai rumusan tertentu
- Sampai pada suatu waktu, umur partikel tsb habis dan menghilang

Air Mancur

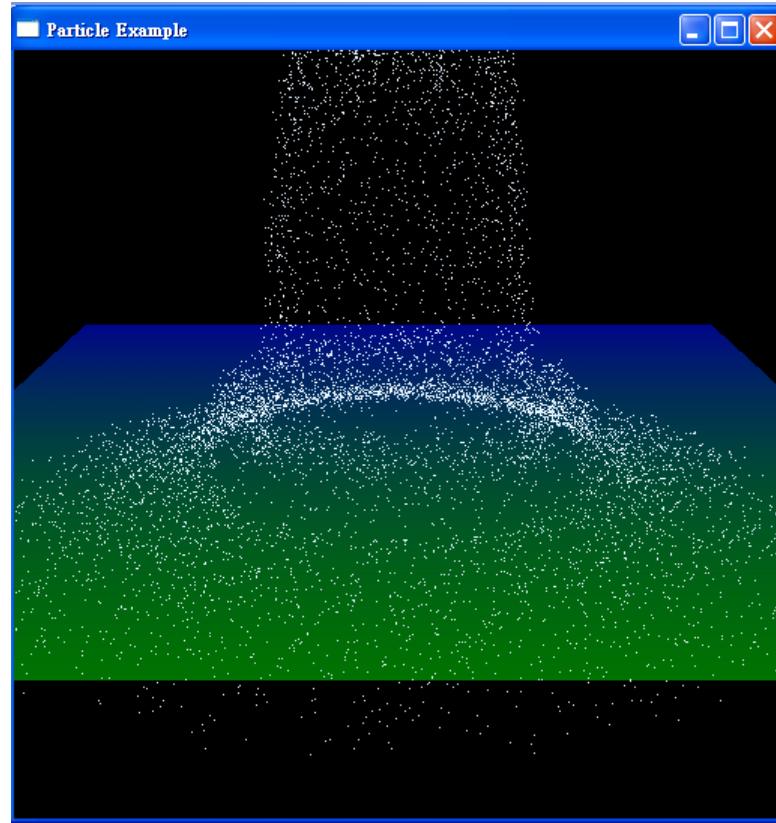
- Inisialisasi partikel

```
void CreateParticle(int i) {  
    particle[i].lifetime= (float)random(500000)/500000.0;  
    particle[i].decay=0.001;  
    particle[i].r = 0.7; particle[i].g = 0.7;  
    particle[i].b = 1.0; particle[i].xpos= 0.0;  
    particle[i].ypos= 0.0; particle[i].zpos= 0.0;  
    particle[i].xspeed = 0.0005-  
        (float)random(100)/100000.0; particle[i].yspeed = 0.01-  
        (float)random(100)/100000.0; particle[i].zspeed =  
        0.0005-(float)random(100)/100000.0; particle[i].active =
```

Air Mancur

- Evolusi hidup partikel

```
void EvolveParticle()
{
    for(int i=0;i<=maxparticle;i++) //untuk setiap particle
    { //umur particle berkurang selama hidupnya sampai
        menghilang
        particle[i].lifetime-=particle[i].decay;
        //menentukan posisi partikel bdsk rumusan geraknya
        particle[i].xpos+=particle[i].xspeed;
        particle[i].ypos+=particle[i].yspeed;
        particle[i].zpos+=particle[i].zspeed;
        particle[i].yspeed-=0.00007;
    }
}
```



Contoh Bezier Curve

- Diberikan suatu kurva bezier derajat 1 dengan titik awal (-2,4) dan titik akhir (-2,4) dan titik kontrolnya adalah (2,2).
- Buatlah plotting titik pada kurva sebanyak 6 titik (termasuk titik awal dan akhir).



Terima Kasih

**Auzi Asfarian, M Asyhar Agmalaro,
Endang P Giri, Shervie N Neyman**

Departemen Ilmu Komputer, FMIPA
IPB University (Institut Pertanian Bogor), Bogor, Indonesia
asfarian@apps.ipb.ac.id | ux@apps.ipb.ac.id



**KOM1304 Grafika Komputer
dan Visualisasi**

Lighting

**Auzi Asfarian, M Asyhar Agmalaro,
Endang P Giri, Shervie N Neyman**

Departemen Ilmu Komputer, FMIPA
IPB University (Institut Pertanian Bogor), Bogor, Indonesia
asfarian@apps.ipb.ac.id | ux@apps.ipb.ac.id



IPB University
— Bogor Indonesia —

Departemen
Ilmu Komputer

Bagian 1: Lighting

Lighting

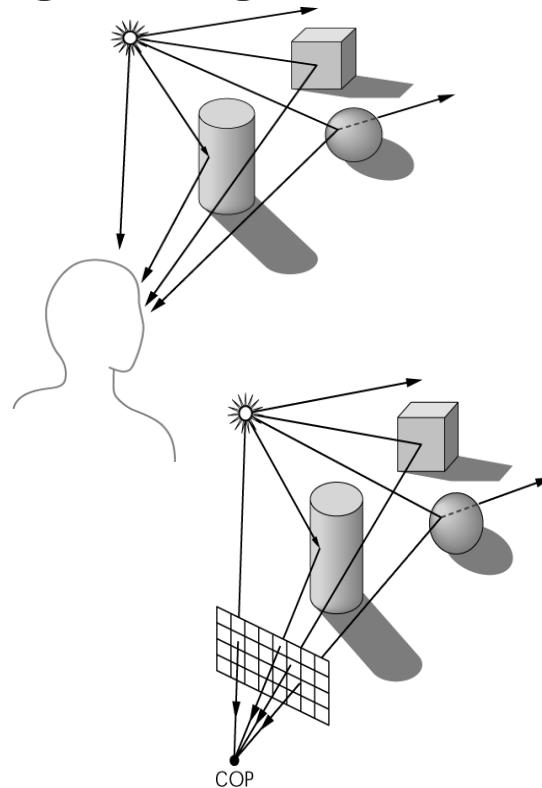
- Rendering
- Light source
- Reflection models
- Shading models

Rendering

- Concerned with determining the most appropriate colour (i.e. RGB tuple) to assign to a pixel associated with an object in a scene
- We need to know
 - how to describe light sources
 - how light interacts with materials - reflection models
 - how to calculate the intensity of light that we see at a given point on object surface - shading models

A Model for Lighting

- Only light that reaches the viewer's eye is ever seen
- Direct light is seen as the colour of the light source
- Indirect light depends on interaction properties
- In computer graphics we replace viewer with projection plane
- Rays which reach COP after passing through viewing plane are actually seen



Illumination Variables

- Light source
 - Positions
 - Properties
- Object
 - Geometry of the object at that point (normal direction)
 - Material properties
 - opaque/ transparent, shiny/dull, texture surface patterns

Modelling the Light Source

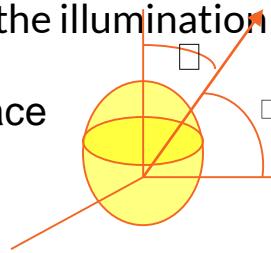
$$I(x, y, z, \theta, \phi, \lambda)$$

- Light sources characterized by the illumination function:

(x, y, z) : point on light source surface

(θ, ϕ) : direction of emission

λ : wavelength



- Contribution of a light source can be determined by integrating over the surface of the light source
- For real-time speeds it is easier if we can approximate with point source (or set of point sources)

Light sources

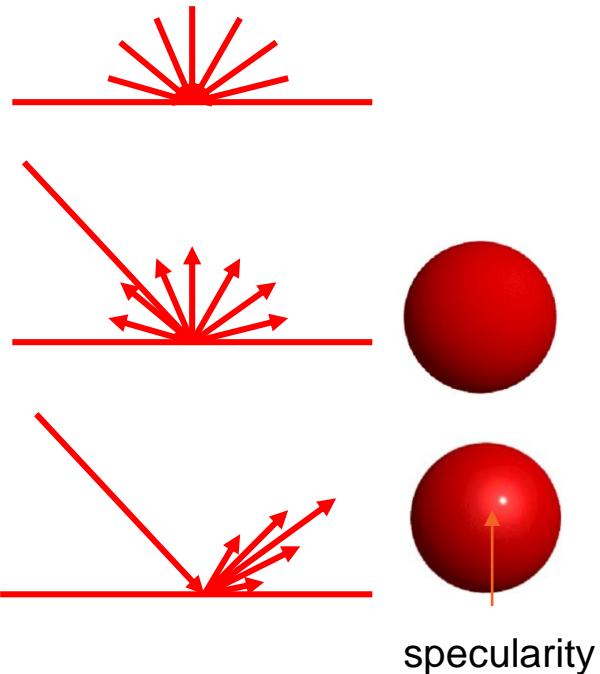
- Ambient light – uniform lighting
- Point source – emits light equally in all directions
- Spotlight – characterized by a narrow range of angles through which light is emitted
- Distance light sources – parallel rays of light

Reflection Model

- A reflection model (also called lighting or illumination model) describes the interaction between light and a surface
- The nature of interaction is determined by the material property
- Three general types of interaction: specular reflection, diffuse reflection, transmission

Reflection Model

- **Ambient** - reflected from other surfaces
- **Diffuse** - from a point source reflected equally in all directions
- **Specular** - from a point source reflected in a mirror-like fashion



The Phong Reflection Model

- The Phong Illumination Model is a local illumination model and is largely an empirical model. However it is fast to compute and gives reasonably realistic results.
- Light incident upon a surface may be reflected from a surface in two ways:
 - **Diffuse reflection:** Light incident on the surface is reflected equally in all directions and is attenuated by an amount dependent upon the physical properties of the surface. Since light is reflected equally in all directions the perceived illumination of the surface is not dependent on the position of the observer. Diffuse reflection models the light reflecting properties of matt surfaces.
 - **Specular Reflection:** Light is reflected mainly in the direction of the reflected ray and is attenuated by an amount dependent upon the physical properties of the surface. Since the light reflected from the surface is mainly in the direction of the reflected ray the position of the observer determines the perceived illumination of the surface. Specular reflection models the light reflecting properties of shiny or mirror-like surfaces.

The Phong Reflection Model

- A local illumination model including only contributions from **diffuse** and **specular** components suffers from one large drawback, namely, a surface that does not have light incident on it will reflect no light and will therefore appear black.
- This is not realistic, for example a sphere with a light source above it will have its lower half not illuminated. In practice in a real scene this lower half would be partially illuminated by light that had been reflected from other objects. This effect is approximated in a local illumination model by adding a term to approximate this general light which is 'bouncing' around the scene. This term is called the **ambient reflection** term and is modelled by a constant term. Again the amount of ambient light reflected is dependent on the properties of the surface.
- Hence the local illumination model that is generally used is

$$\text{illumination} = \text{Ambient} + \text{Diffuse} + \text{Specular}$$

Ambient Reflection

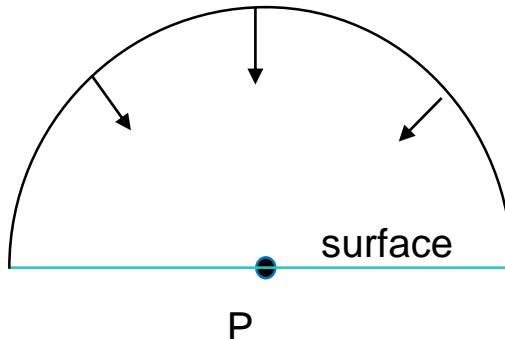
In the Phong model ambient light is assumed to have a constant intensity throughout the scene. Each surface, depending on its physical properties, has a coefficient of ambient reflection which measures what fraction of this light is reflected from the surface. Hence for an individual surface the intensity of ambient light reflected is:

$$I = K_a L_a$$

I = Reflected intensity

K_a = Reflection coefficient

L_a = Ambient light intensity (same at every point)

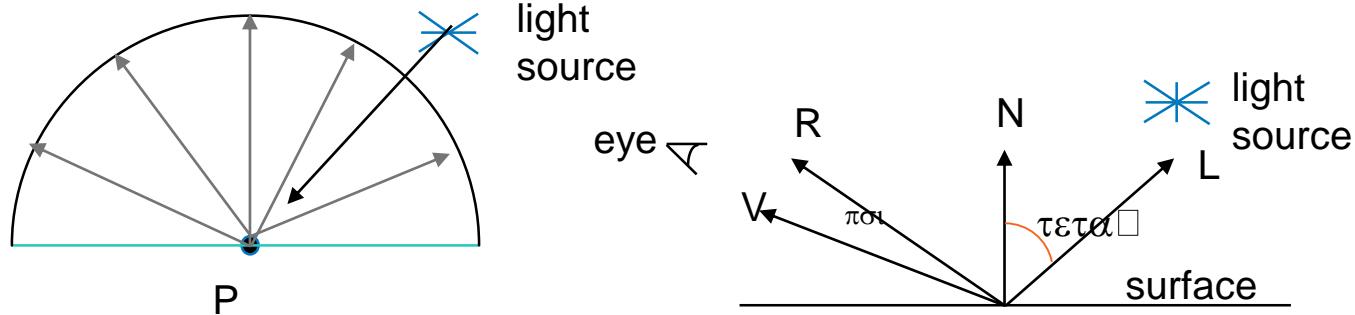


Diffuse Reflection

A perfectly diffuse reflecting surface scatters light equally in all directions. Thus the intensity at a point on a surface as perceived by the viewer does not depend on the position of the viewer.

The colour of the light reflected from the surface depends upon the colour of the light and the properties of the surface. Light incident on the surface will have some components absorbed and others scattered thus giving the surface its colour. Thus a surface that appears red under white light absorbs green and blue and scatters red light. When only diffuse light is considered surfaces will appear dull or matt.

Diffuse Reflection



The intensity due to diffuse reflection is given by
Lambert's cosine law:

$$I_d = K_d I_i \cos \tau\alpha$$

I_d = Reflected intensity

K_d = Diffuse reflection coefficient

I_i = Intensity of the incident light

If there is more than one light source then the diffuse intensity is summed over all light sources.



The Effect of Distance

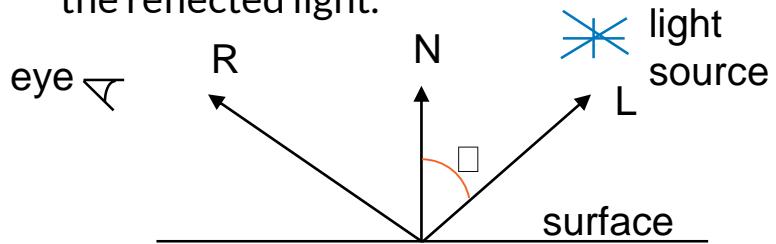
With the lighting model proposed so far two surfaces with the same properties and orientation but different distances from the light source would have the same intensity of illumination. This can be corrected by including a factor dependent on the *distance of the surface point to the viewing point*. Hence the lighting model (for a single light source) and modelling only ambient and diffuse reflection is now:

$$I = K_a I_a + \frac{K_d I_i \cos\theta}{r + k}$$

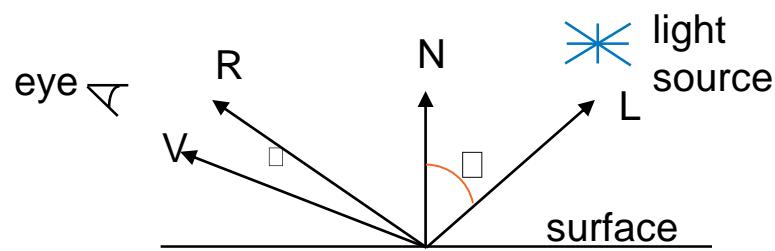
where r is the distance and k is an arbitrary constant chosen to make the image appear correct.

Specular Reflection

- Specular reflection is caused by the mirror-like properties of a surface. A perfect mirror will reflect light arriving at the surface at an angle of incidence *theta* to the normal at a reflected angle of *theta* to the normal in the same plane as the normal and the incident light. This means that only a viewer on the reflected ray will actually see the reflected light.



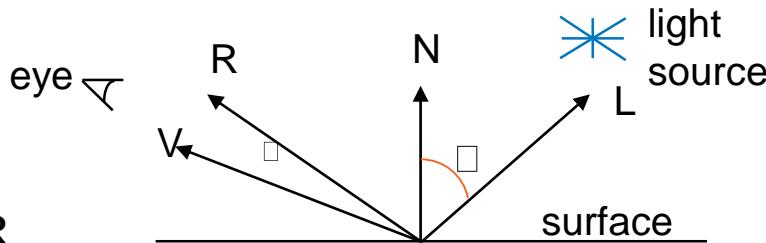
Specular Reflection



- In practice no surface is a perfect mirror and there will be a certain amount of light scattered around the reflected direction.
- The reflected light is therefore seen over an area of the surface as a **highlight**. The colour of this specularly reflected highlight is usually taken to be that of the light source. In diffuse reflection the reflected light is the colour of the surface.
- In practice the distribution function for specularly reflected light is a complex function of *Phi* - the angle between the reflected ray and the viewing direction **V**.

Specular Reflection

- Amount of light visible to viewer depends on the angle ϕ between R and V:
- $I_s = K_s I_i (\cos \phi)^n$
- n varies with material, large n: shiny, small n: dull



V = Direction to the viewer (COP)

R = Direction of perfect reflected light

N = Surface normal

L = Direction of light source

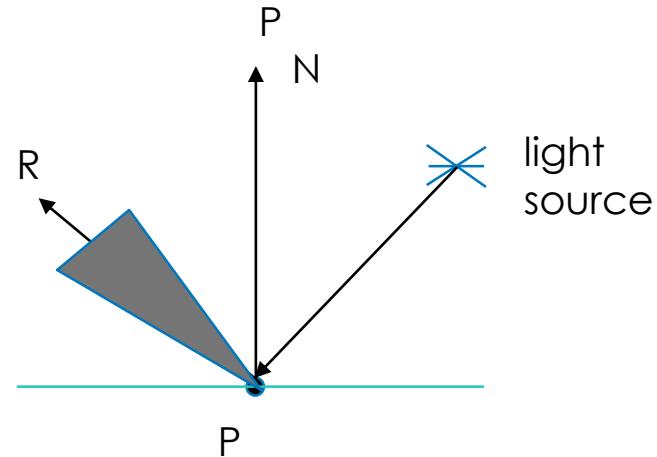
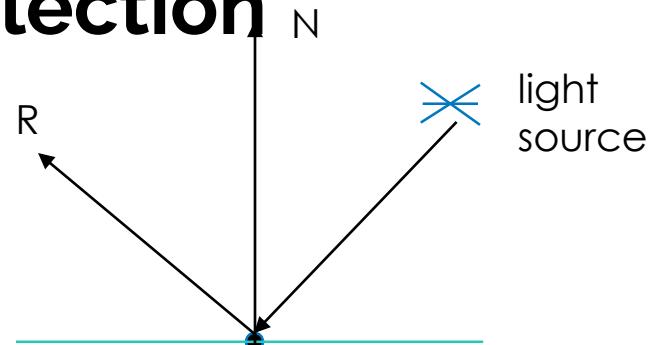
I_i = Intensity of the incident light

K_s = Specular reflection coefficient

I_s = Reflected intensity

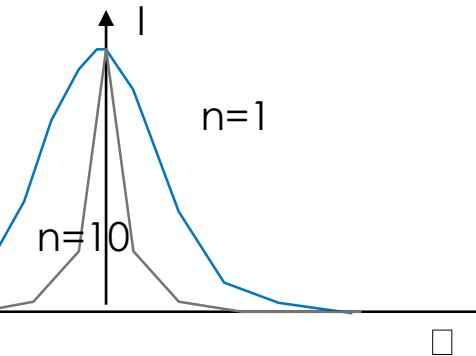
Specular Reflection

- In perfect specular reflection, light is reflected along the direction symmetric to the incoming light
- In practice, light is reflected within a small angle of the perfect reflection direction - the intensity of the reflection tails off at the outside of the cone. This gives a narrow highlight for shiny surfaces, and a broad highlight for dull surfaces



Specular Reflection

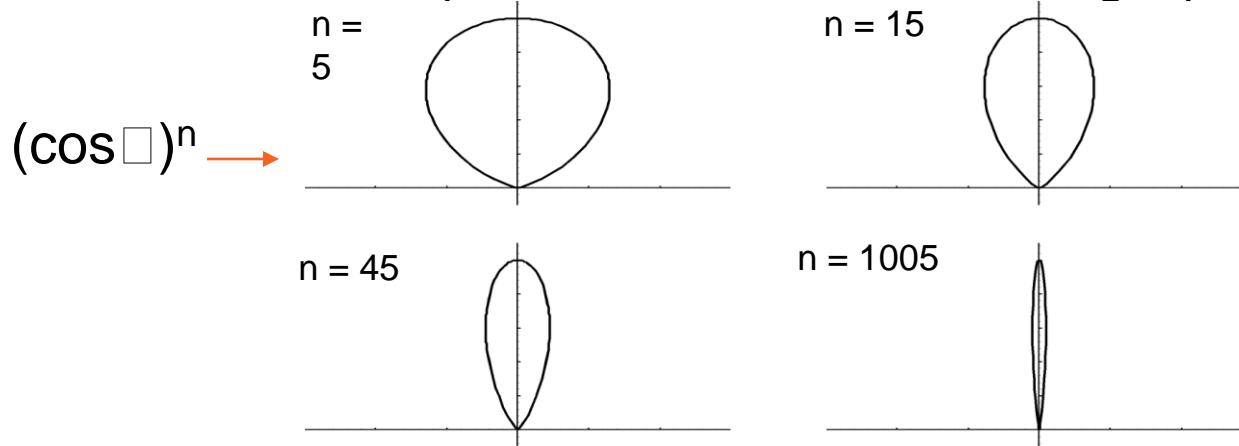
- Thus we want to model intensity, I , as a function of angle between viewing direction and the reflection, say θ , with a sharper peak for shinier surfaces, and broader peak for dull surfaces



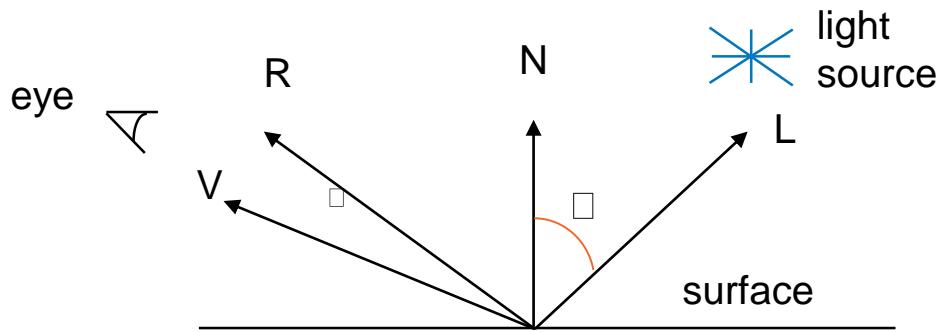
- This effect can be modelled by $(\cos \theta)^n$, with a sharper peak for larger n
- Empirical

Specular Highlights

- The cosine function (defined on the sphere) gives us a lobe shape which approximates the distribution of energy about a reflected direction controlled by the shininess parameter n known as the Phong exponent.



Ambient, Diffuse and Specular



Hence we obtain the complete basic reflection model:

$$I = K_a I_a + \frac{K_d I_i (L \bullet N) + K_s I_i (R \bullet V)^n}{r + k}$$

Note that it is assumed that N, R and V are unit vectors

The coefficient of Specular Reflection is treated as a constant in the Phong model. However it is actually a function of the angle of incidence.

Colour

- To handle colour the usual approach is to treat specular highlights as being the same colour as the light source.
- The colour of objects is handled by treating the coefficients of ambient and diffuse reflection as having red, green and blue components. Hence the red, green and blue signals that drive the display are produced from:

$$I_r = K_{ar} I_a + \frac{K_{dr} I_i (L \bullet N) + K_s I_i (R \bullet V)^n}{r + k}$$

$$I_g = K_{ag} I_a + \frac{K_{dg} I_i (L \bullet N) + K_s I_i (R \bullet V)^n}{r + k}$$

$$I_b = K_{ab} I_a + \frac{K_{db} I_i (L \bullet N) + K_s I_i (R \bullet V)^n}{r + k}$$

The Diffuse and Specular terms are summed over each light source

Summary of the Phong Model

- Light sources are usually considered as point sources situated infinitely far away. Hence the angle between the incident light and the normal to a planar surface is constant over a planar surface.
- The viewer is assumed positioned at infinity, hence the angle between the viewing direction and the reflected ray is constant over a planar surface.
- The diffuse and specular terms are modelled as local components only.
- An empirical result is used to model the distribution of the specular term around the reflection vector.
- The colour of the specular term is assumed to be that of the light source.
- The global illumination is modelled as a constant ambient term.
- Shadows are not handled.
- The coefficient of diffuse reflection is wavelength dependent. But it is sampled at each of the three primary colours.
- The coefficient of specular reflection is modelled as a constant but is actually dependent on the angle of incidence.

Local Shading Models

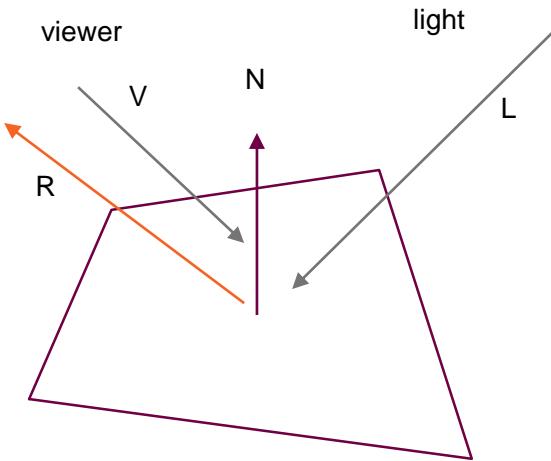
- Local shading models provide a way to determine the intensity and color of a point on a surface
- They do not require knowledge of the entire scene, only the current piece of surface.
- For the moment, assume:
 - We are applying these computations at a particular point on a surface
 - We have a normal vector for that point

Flat Shading

- Compute shading at a representative point and then generalise to the whole polygon
 - For a flat polygon the vector n is constant across the surface
 - If we assume a distant viewer, v is constant
 - If we assume a distant light source l is constant
- If the three vectors are constant then the shading calculation needs to be calculated only once for each polygon

Flat Shading

- Calculate normal
- Assume $L \cdot N$ and $R \cdot V$ constant
(light & viewer at infinity)
- Calculate I_r, I_g, I_b using Phong reflection model
- Project vertices to viewplane
- Use scan line conversion to fill polygon



Flat Shading

- When flat shading is used the same shade is used over the whole surface. If the surface actually corresponds to a planar surface in the model then this is correct. Though remember that as the Phong Illumination Model assumes that the light source is at infinity a large surface illuminated by a light positioned close to the surface will not be evenly illuminated in reality.
- The shade to be associated with a surface is calculated before View Transformation and Perspective Transformation are carried out. That is before all the angles are distorted.
- If flat shading of a planar surface is to be considered then an arbitrary point on the surface can be chosen at which to calculate the shade.
- If a curved surface is approximated by a mesh of polygonal facets then two approaches to calculating the shade of the facets can be used:
- Calculate the shade at some point on each facet, for example its centre.
- Calculate the shade at the vertices of the mesh using true surface normals. The shade at each facet can then be evaluated as the average of the shades at its vertices.
- Flat shading provides realistic images of objects which are composed entirely of planar surfaces but produce a faceted appearance when used to approximate curved surfaces.

2D Graphics - Filling a Polygon

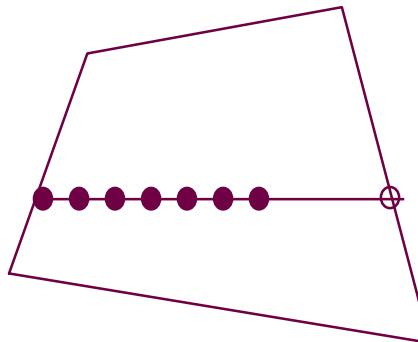
- Scan line methods used to fill 2D polygons with a constant colour

- find y_{min} , y_{max} of vertices

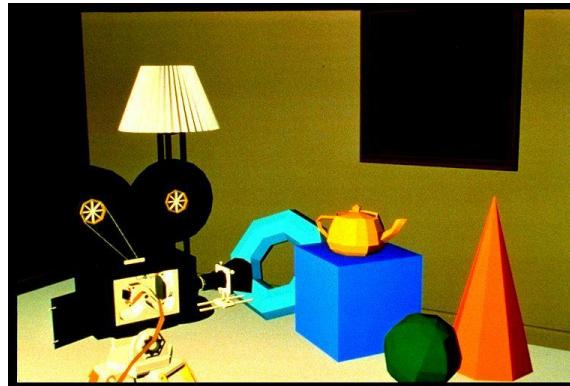
- from y_{min} to y_{max} do:

- find intersection with polygon edges

- fill in pixels between intersections using



Flat Shading



Smooth Shading

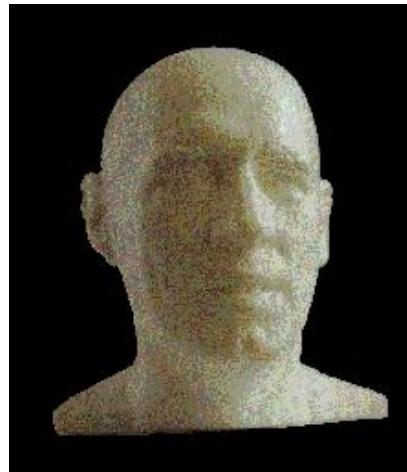
- In many cases the polygons that are to be rendered in the polygon pipeline are an approximation to a curved surface.
- In this case the different shade calculated on each planar polygon will give a faceted appearance to the surface and it will not look like a smooth curved surface at all.
- To avoid this problem **incremental, or smooth, shading methods** are used to produce smoothly shaded pictures from planar polygon faceted representations of curved surfaces.

Smooth Shading

- The first step is to produce a shade value at each vertex of the polygonal mesh representing the surface. This can be done in two ways:
 1. Evaluate a surface normal on each polygonal facet. Produce a surface normal at each vertex by averaging the surface normals for the surrounding facets. The shade at the vertex can now be calculated.
 2. Evaluate the shade directly at each vertex by considering the vertex as a point on the actual curved surface and evaluating the actual surface normal to the surface at that point.
- Once the shade at the vertices of the polygonal mesh are known the shade at points interior to the polygonal facets are interpolated from the values at the vertices.
- This technique makes curved surfaces look 'smooth shaded' even though based on a planar facet representation. The interpolation of shade values is incorporated into the polygon scan conversion routine.
- Hence an increase in realism is obtained at far less expense in computation than carrying out a pixel-by-pixel shading calculation over the whole original surface.

Gouraud Shading

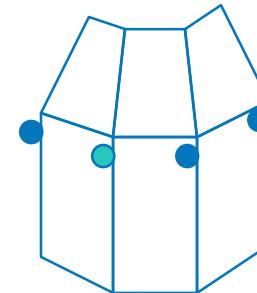
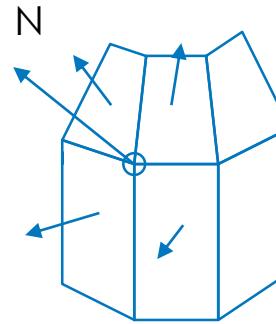
- Gouraud shading is a method for linearly interpolating a colour or shade across a polygon. It was invented by Gouraud in 1971
- It is a very simple and effective method of adding a curved feel to a polygon that would otherwise appear flat
- It can also be used to depth queue a scene, giving the appearance of objects in the distance becoming obscured by mist



Henri Gouraud is another pioneering figure in computer graphics

Gouraud Shading

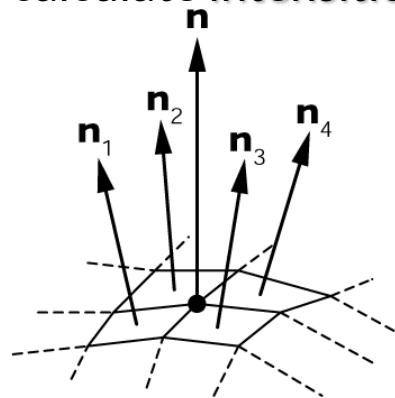
- Gouraud shading attempts to smooth out the shading across the polygons
- Unlike a flat shaded polygon, it specifies a different shade for each vertex of a polygon
- The rendering engine then smoothly interpolates the shade across the surface



Gouraud Shading

- Begin by calculating the normal at each vertex
- A feasible way to do this is by averaging the normals from surrounding polygons
- Then apply the reflection model to calculate intensities at each vertex

$$\mathbf{n} = \frac{\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4}{|\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4|}$$

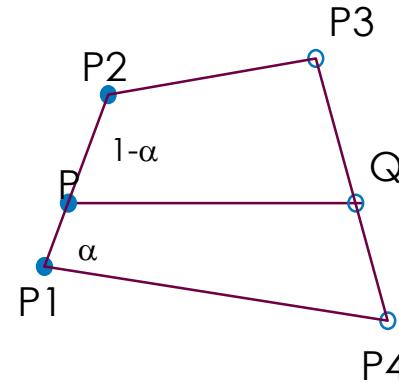


Gouraud Shading

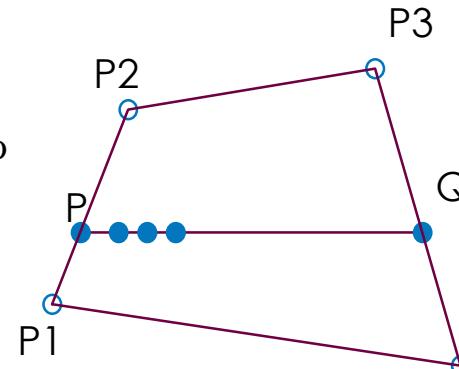
- We use linear interpolation to calculate intensity at edge intersection P

$$I_P^{\text{RED}} = (1-\alpha)I_{P1}^{\text{RED}} + \alpha I_{P2}^{\text{RED}}$$

where P divides P1P2 in the ratio $\boxed{\text{?}}:\boxed{\text{?}}:\boxed{\text{?}}:1-\boxed{\text{?}}$

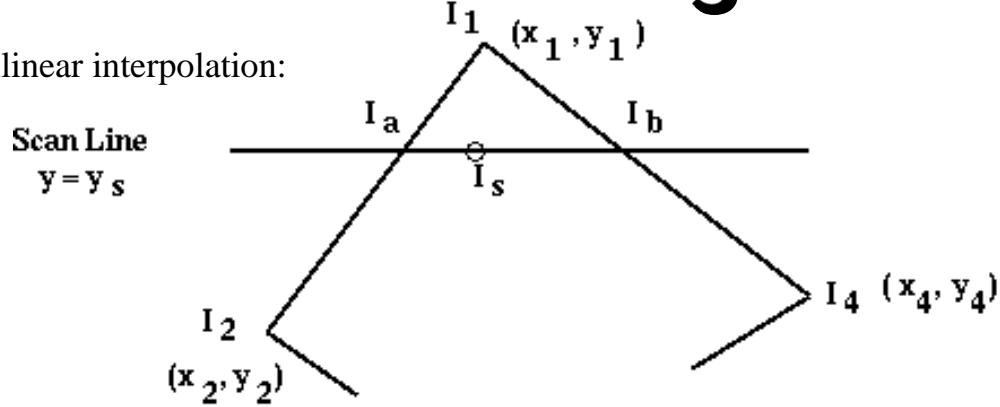


- Similarly for Q
- Then we do further linear interpolation to calculate colour of pixels on scanline PQ



Gouraud Shading

- Bilinear interpolation:



$$I_a = \frac{1}{y_1 - y_2} (I_1(y_s - y_2) + I_2(y_1 - y_s))$$

$$I_b = \frac{1}{y_1 - y_4} (I_1(y_s - y_4) + I_4(y_1 - y_s))$$

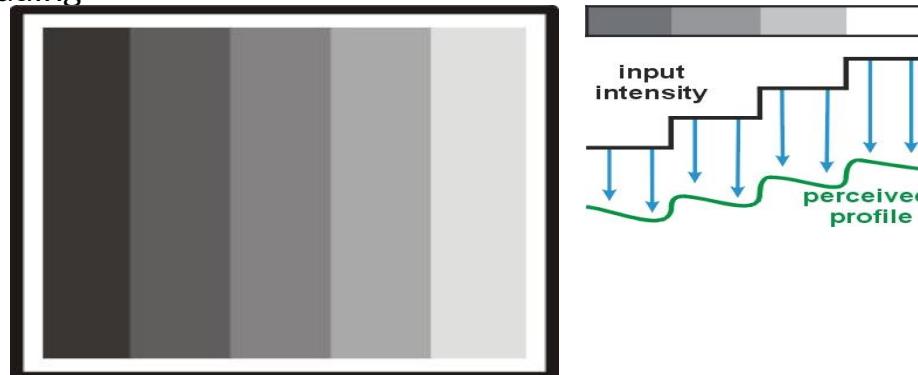
$$I_s = \frac{1}{x_b - x_a} (I_a(x_b - x_s) + I_b(x_s - x_a))$$

Gouraud Shading Limitations

- Constant shading is simple, but it emphasizes the polygonal nature of the model. Our perceptual system exaggerates the effect because we see very clearly the boundary between adjacent light and dark shades (Mach banding)
- Problems with Gouraud shading occur when you work with large polygons. A specular highlight at a vertex tends to be smoothed out over a larger area than it should cover - imagine you have a large polygon, lit by a light near its center. The light intensity at each vertex will be quite low, because they are far from the light. The polygon will be rendered quite dark, but this is wrong, because its centre should be brightly lit
- Gouraud shading can make a real difference over flat shaded polygons

Gouraud Limitations - Mach Bands

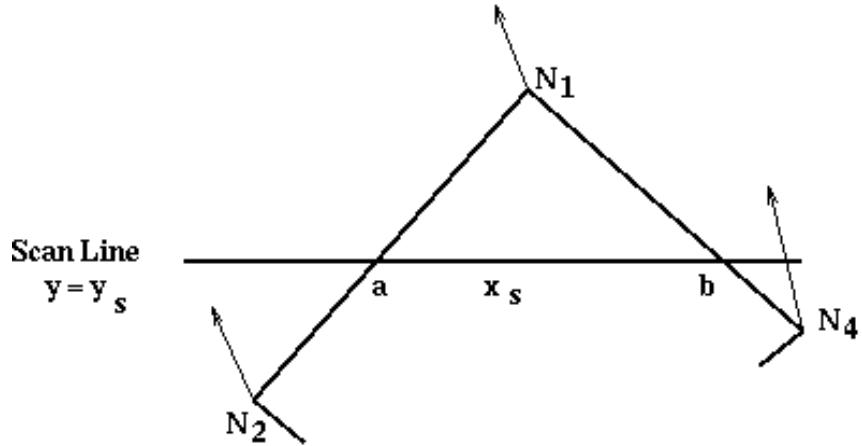
- The rate of change of pixel intensity is even across any polygon, but changes as boundaries are crossed
- This 'discontinuity' is accentuated by the human visual system, so that we see either light or dark lines at the polygon edges - known as Mach banding
- Mach bands where the intensity is constant in bands – similar effect with Gouraud shading



Phong Shading

- Phong shading uses similar principles to Gouraud shading but instead of interpolating the shade across the surface it is the **surface normals that are interpolated**. The interpolated normals are then used to evaluate a shade at each pixel.
- As before the light source and the viewer are assumed to be at infinity so that the intensity at a point is a function only of the interpolated normal.
- The equations for the interpolated normals are similar to Gouraud shading but each component of the normal has to be interpolated and the vector re-normalised after each interpolation step.

Phong Shading



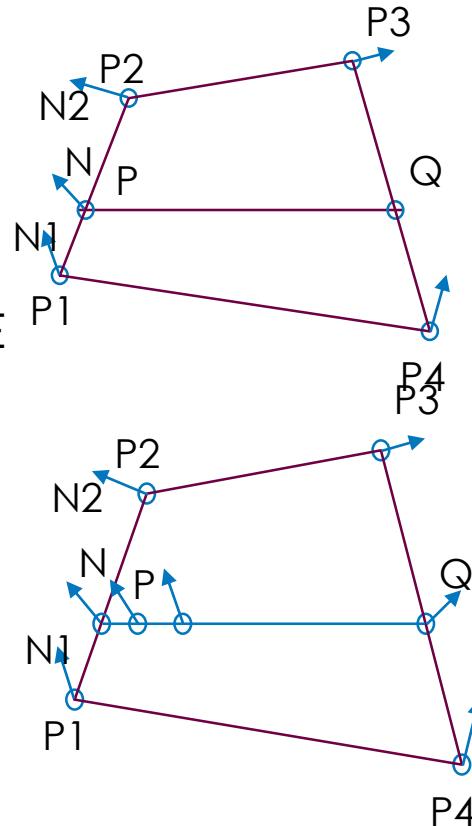
$$N_a = \frac{1}{y_1 - y_2} (N_1(y_s - y_2) + N_2(y_1 - y_s))$$

$$N_b = \frac{1}{y_1 - y_4} (N_1(y_s - y_4) + N_4(y_1 - y_s))$$

$$N_s = \frac{1}{x_b - x_a} (N_a(x_b - x_s) + N_b(x_s - x_a))$$

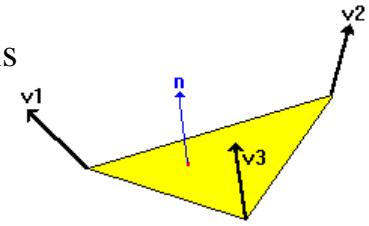
Phong Shading

- Phong shading interpolates normals at each pixel, then apply the reflection model at each pixel to calculate the intensity IRED, IGREEN, IBLUE (shade each pixel individually)
- Every single pixel has its brightness calculated using the interpolated normal vector



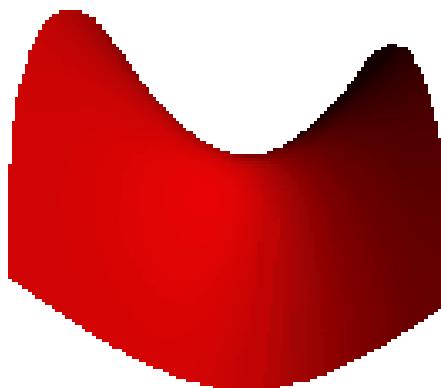
Phong Shading

- For a pixel on this polygon, the fraction of light from the light source reflecting off this pixel is the dot product of n and I .
- The dot product function returns values in the range 1 to -1. 1 is full brightness.
- There is no such thing as negative light, so values less than 0 should be taken as 0.
- If you multiply this value by the brightness of the light, then you have the brightness of that pixel

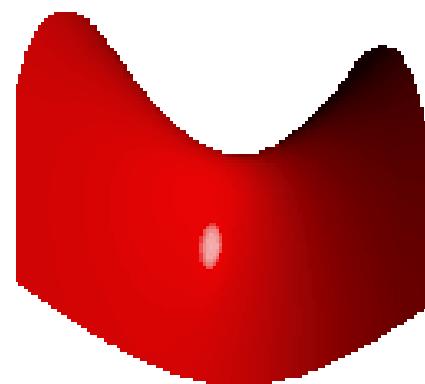


Gouraud versus Phong

Gouraud



Phong



Phong versus Gouraud Shading

- A major advantage of Phong shading over Gouraud is that specular highlights tend to be much more accurate, vertex highlight is much sharper
- The cost is a substantial increase in processing time because reflection model applied per pixel
- But there are limitations to both Gouraud and Phong
- In practice, some simplifications are made to the model for efficiency. For example, ambient light is sometimes assumed to be a constant

Flat, Gouraud and Phong Shading



Shading Summary

- It is expensive to calculate the intensity at each pixel of a projected polygon, and so there is a family of methods which use interpolation to calculate intermediate values
 - constant shading: reflection calculation is carried out once for each polygon, and the polygon is assigned a constant shade according to that calculation
 - Gouraud shading: an estimate is made of the normal at each vertex of a polygon (for example, by averaging the normals of all surrounding polygons); reflected intensity at each vertex is then calculated; linear interpolation is then used to estimate the intensity at any interior pixel as the shading is carried out
 - Phong shading: as with Gouraud, normal estimates are made at each vertex; but in contrast, we proceed to interpolate the normals at each pixel and then calculate the intensity for that pixel



Terima Kasih

**Auzi Asfarian, M Asyhar Agmalaro,
Endang P Giri, Shervie N Neyman**

Departemen Ilmu Komputer, FMIPA
IPB University (Institut Pertanian Bogor), Bogor, Indonesia
asfarian@apps.ipb.ac.id | ux@apps.ipb.ac.id



KOM1304 Grafika Komputer dan Visualisasi

Interaktivitas

**Auzi Asfarian, M Asyhar Agmalaro,
Endang P Giri, Shervie N Neyman**

Departemen Ilmu Komputer, FMIPA
IPB University (Institut Pertanian Bogor), Bogor, Indonesia
asfarian@apps.ipb.ac.id | ux@apps.ipb.ac.id



IPB University
— Bogor Indonesia —

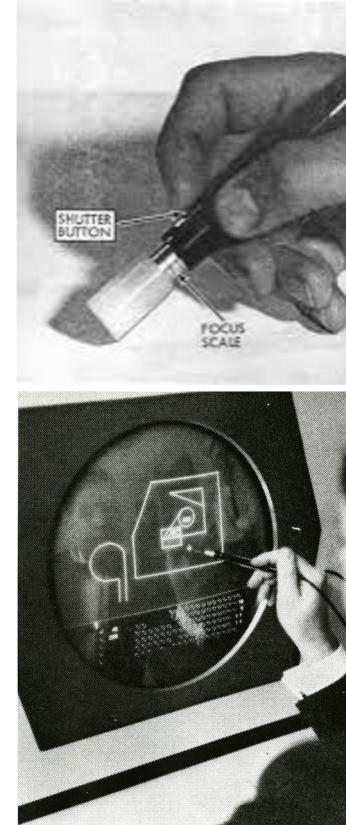
Departemen
Ilmu Komputer

Bagian 1: Perangkat Interaksi



Shuterland IE. 1963. *Sketchpad*, a man-machine graphical communication system (Disertasi). Massachusetts (US): MIT.

https://www.youtube.com/watch?v=6orsmFndx_o



Interaksi Manusia-Komputer

Dimungkinkannya interaksi antara manusia dengan perangkat *display* merupakan suatu terobosan terpenting dalam teknologi komputer.

Sejak dipopulerkan oleh Sketchpad (Sutherland 1963), telah lahir banyak terobosan perangkat yang menghubungkan manusia dengan komputer.

<https://www.youtube.com/watch?v=5RyU50qbvzQ>

Pointing Devices

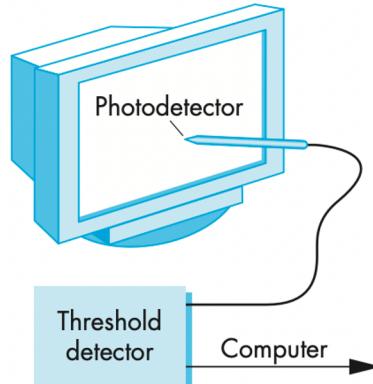


FIGURE 3.2 Light pen.



FIGURE 3.3 Mouse.

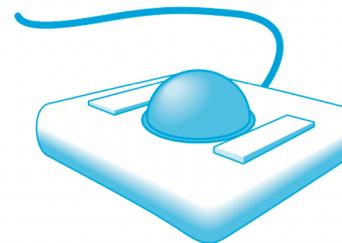


FIGURE 3.4 Trackball.

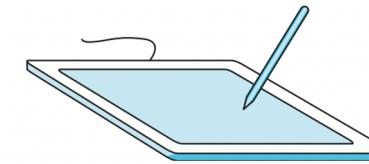


FIGURE 3.6 Data tablet.

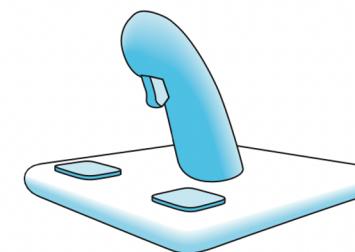


FIGURE 3.7 Joystick.

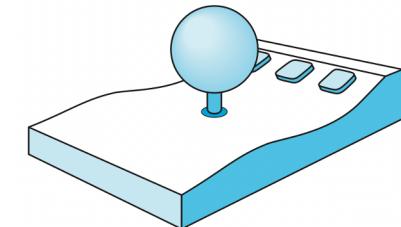
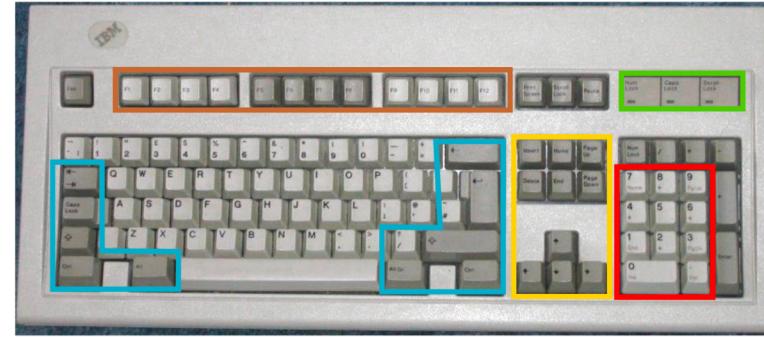


FIGURE 3.8 Spaceball.

Angel E, Shreiner D. Interactive computer graphics with WebGL. Addison-Wesley Professional; 2014 Mar 10.

Typing Device



From Experimental Devices....

<https://www.youtube.com/watch?v=NtwZXGprxag>



Shuterland IE. 1965. *The Ultimate Display*. Proceedings of IFIPS Congress. New York, Mei 1965. Hal. 506-508.

<https://www.youtube.com/watch?v=aKL0urEdtPU>



Cruz-Neira C, Sandin DJ, DeFanti TA, Kenyon RV, Hart JC. The CAVE: audio visual experience automatic virtual environment. Communications of the ACM. 1992 Jun 1;35(6):64-73.

... to Public Availability



Head-Mounted Display

https://www.youtube.com/watch?v=GPahi9_asYk



Singapore Art and Science Museum CAVE-like system

Example of Novel Interaction Devices

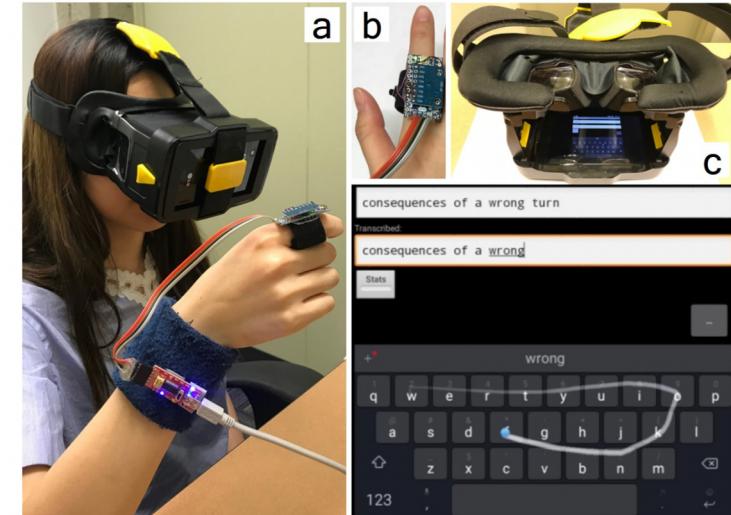
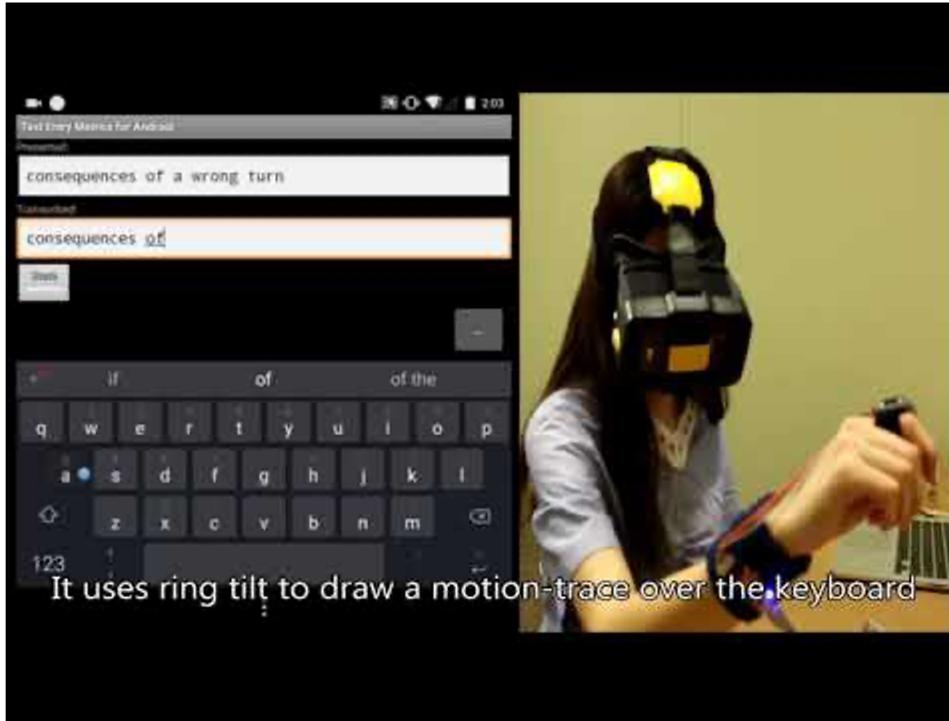
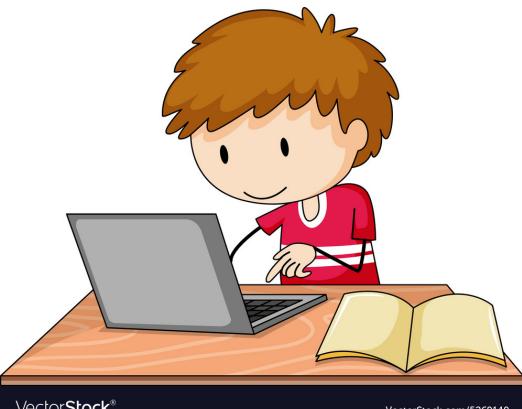


Figure 2: a) User typing using RotoSwype. b) Ring with button. c) MoGo headset with phone inside. d) The phone screen (what the user sees inside the headset). Shows the round, blue pointer drawing a trace over keyboard. The button to the right of letter *m* is designated as the Submit button.

<https://www.youtube.com/watch?v=qyHb4uPV8xw> (menit ke-5:42)

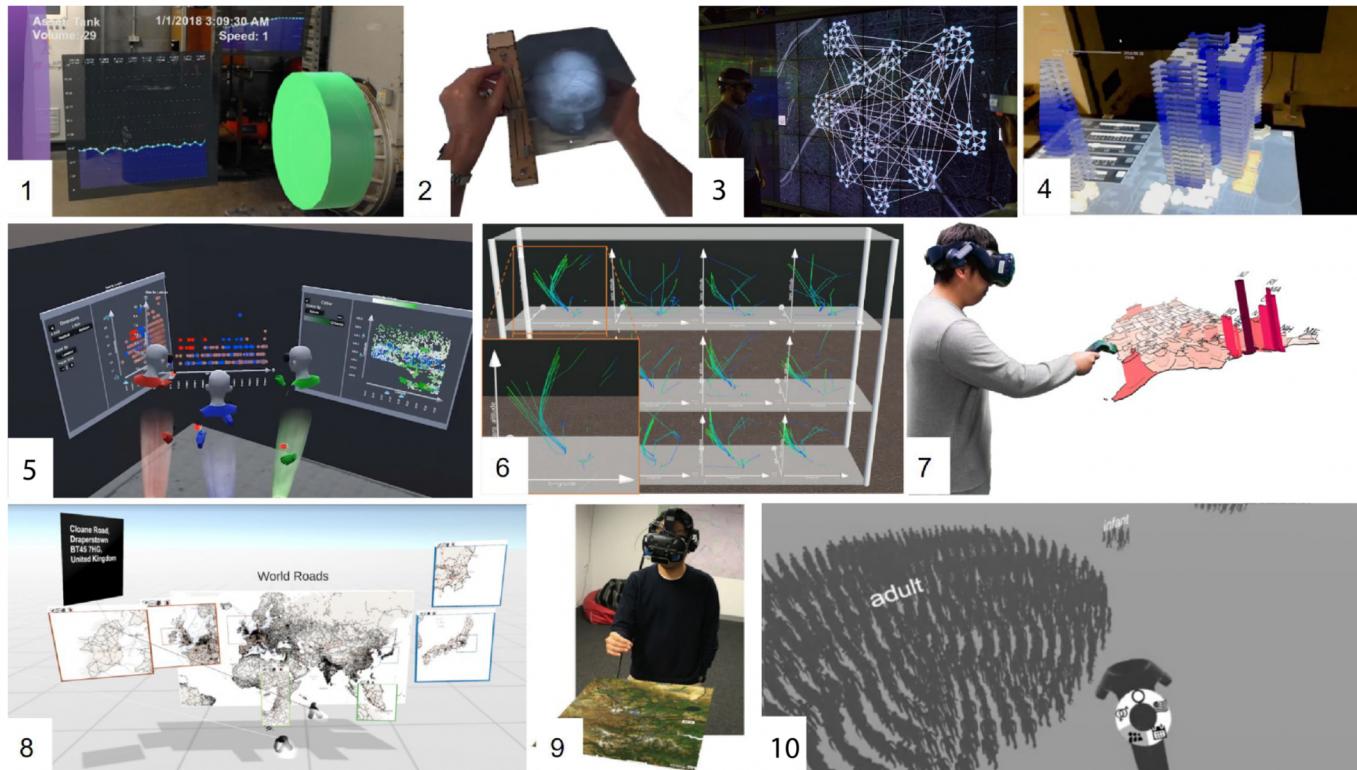
Tugas sekaligus presensi

- https://commons.wikimedia.org/wiki/Category:Computer_keyboards#/media/File:Cybercrime_-_keyboard_and_handcuffs.jpg
- Pilih salah satu jenis keyboard, ambil berdasarkan abjad pertama nama anda, defaultnya “C”
- Buka : <https://ipb.link/kom1304-pertemuan7>
- Isi (Sekarang juga)



Example of Novel Interaction Devices

<https://hal.archives-ouvertes.fr/hal-02614521/document>



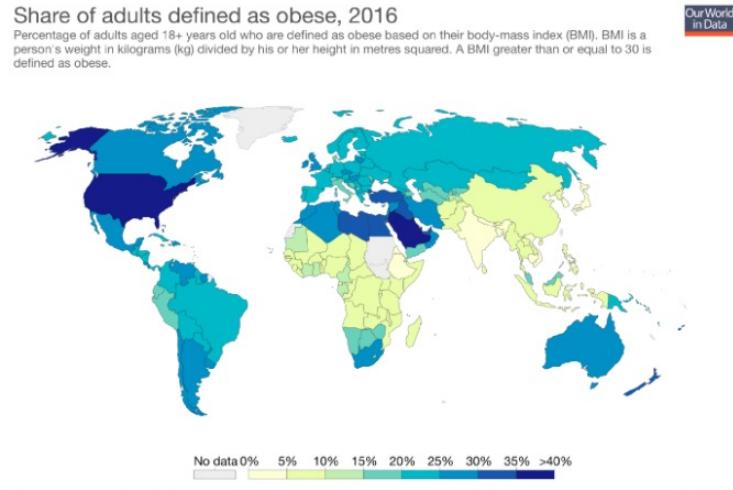
Immersive Analytics

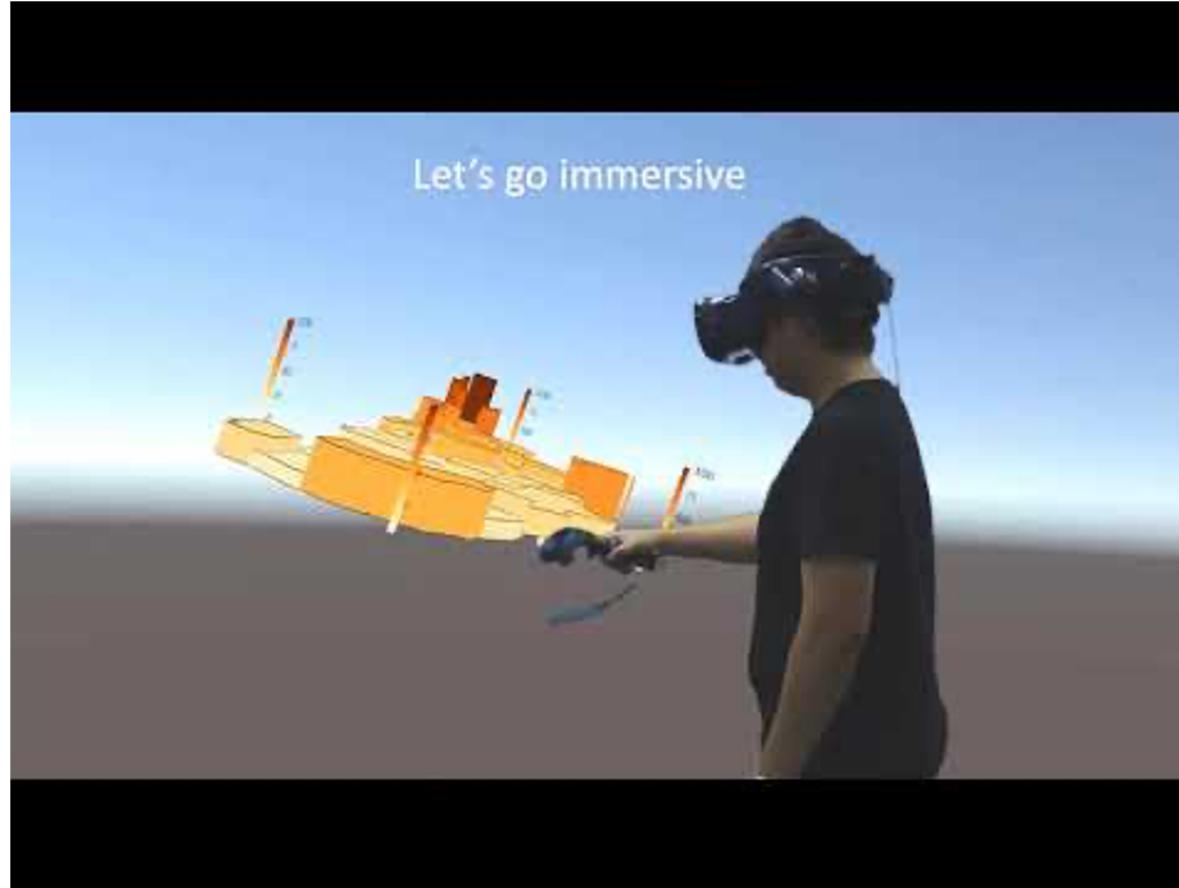
Figure 1: Illustration of several recent Immersive Analytics systems: (1) Corsican Twin, (2) the Embodied Axes controller for precise interaction with 3D visualisations in AR, (3) AR personal navigation of networks on wall-size displays, (4) the UpLift system mixes AR, a touch table and tracked tangibles for building energy analytics (image ©IEEE), (5) the FIESTA system for collaborative immersive analytics in VR, (6) a study of small-multiples in VR, (7) the Tilt Map system for 3D Choropleth maps visualisation (image ©IEEE), (8) a multi-scale navigation system for map navigation in AR, (9) bare hand interactions to navigate AR maps, and (10) immersive unit visualisation

Ens B, Bach B, Cordeil M, Engelke U, Serrano M, Willett W, Prouzeau A, Anthies C, Büschel W, Dunne C, Dwyer T. Grand challenges in immersive analytics. In Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems 2021 May 6 (pp. 1-17)..

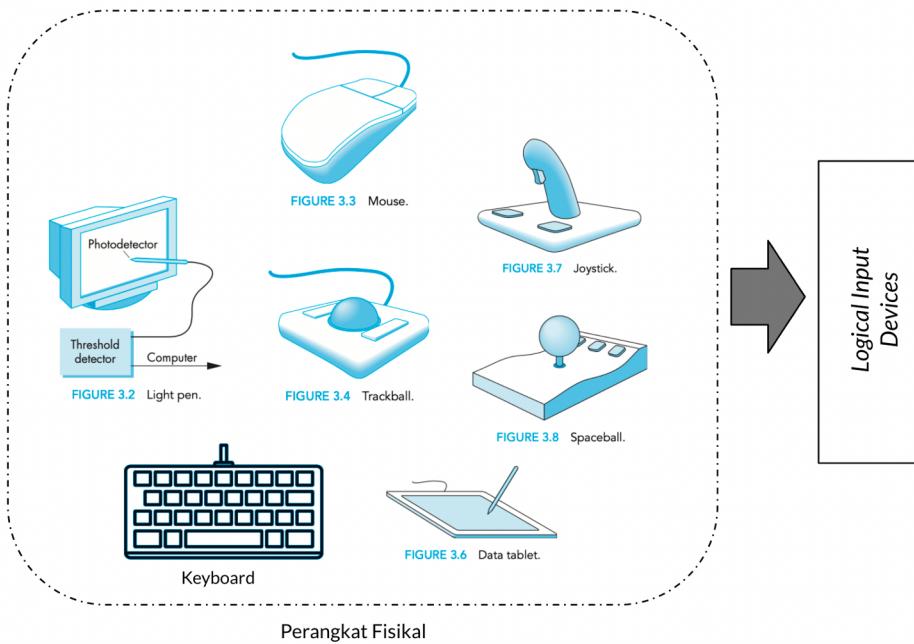
Choropleth map

- Peta Choropleth adalah jenis peta tematik di mana area diarsir atau berpola sebanding dengan variabel statistik (dari Wikipedia). Peta ini misalnya menunjukkan persentase orang dewasa yang mengalami obesitas di tahun 2016.





Perangkat Masukan Logikal



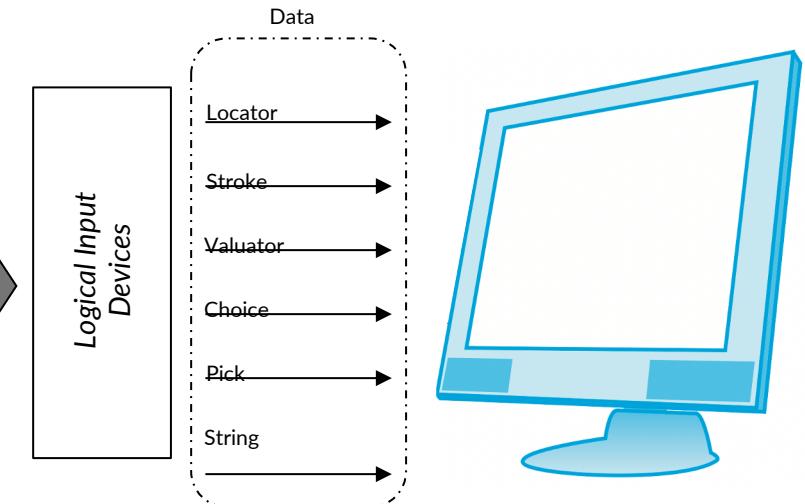
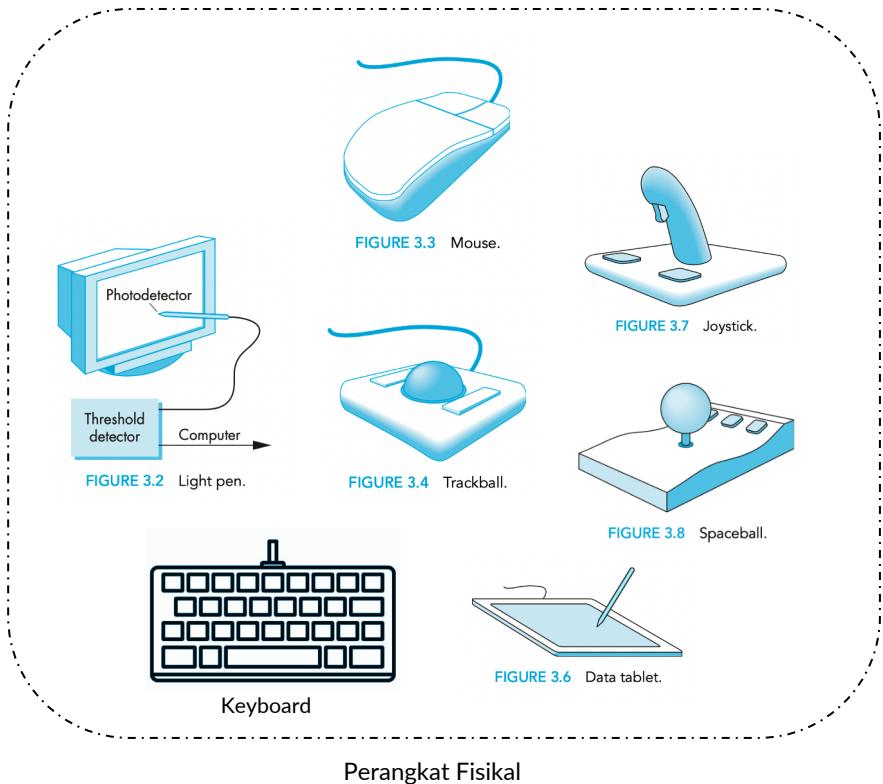
Banyaknya jenis dan merek perangkat masukan mendorong standarisasi untuk meningkatkan kompatibilitas.

Standar untuk grafika komputer diatur pada Graphical Kernel System (GKS) (ISO/IEC 7942-4:1998) yang diperluas untuk mencakup kapabilitas 3d di GKS-3D (ISO 8805:1988)

Salah satu yang diatur pada GKS adalah definisi enam perangkat masukan logikal beserta data yang dikirimnya ke suatu sistem komputer.

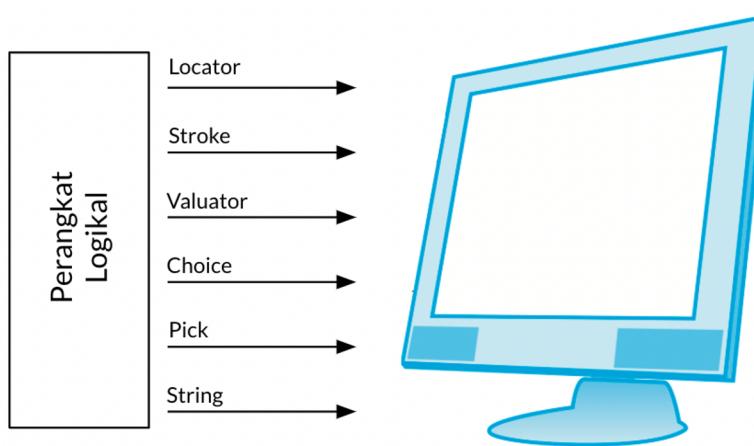
Standar ini digunakan baik oleh pengembang perangkat keras maupun di sisi perangkat lunak. Memungkinkan interoperabilitas perangkat.

Dari Fisikal ke Logikal



Hopgood, F. R. A. (1983). Introduction to the Graphical Kernel System (GKS). London: Academic Press. ISBN 0-12-355570-1.
Angel E, Shreiner D. Interactive computer graphics with WebGL. Addison-Wesley Professional; 2014 Mar 10.
www.iso.org/obp/ui/es/#iso:std:iso:8805:ed-1:v1:en
<https://www.iso.org/obp/ui/#iso:std:iso-iec:7942:-4:ed-1:v1:en>

Enam Jenis Data Masukan Logikal GKS



String: Data ASCII untuk mewakili suatu karakter atau simbol yang dapat dipahami oleh manusia.

Locator: Posisi relatif pada *world coordinate* (sistem koordinat kartesian yang digunakan oleh program untuk menspesifikasi input dan output grafis).

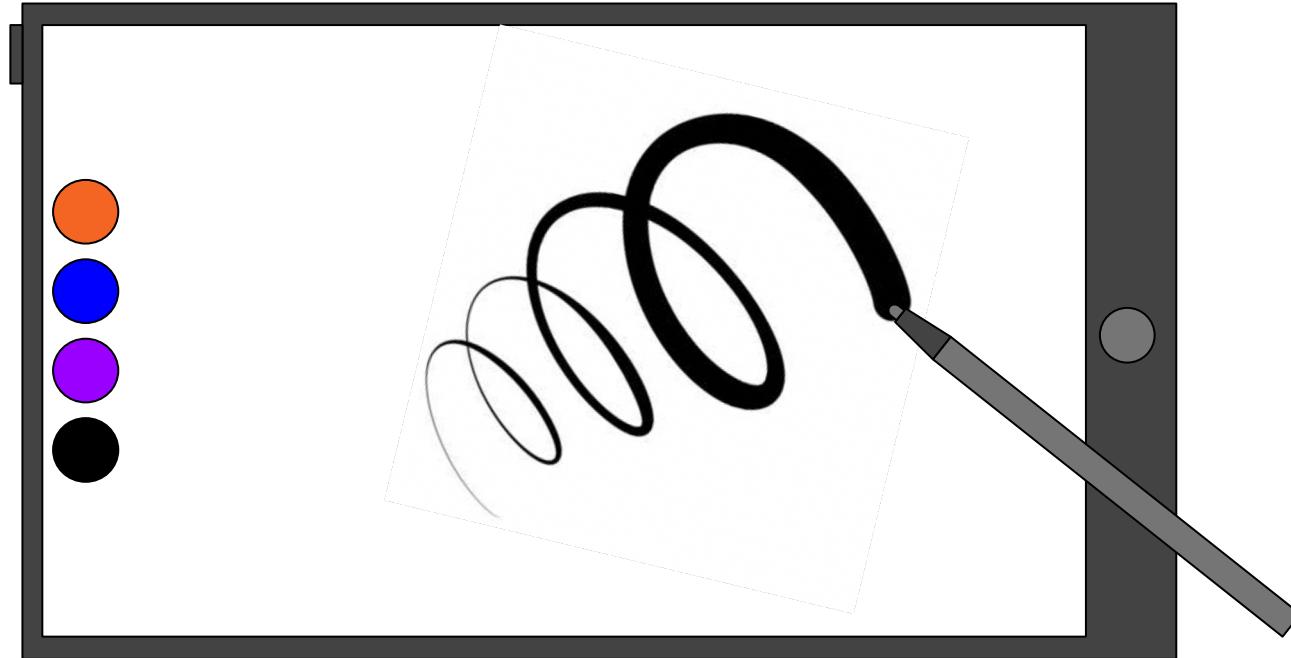
Stroke: Serangkaian informasi *locator*.

Valuator: Suatu nilai seperti bilangan *real*.

Choice: Pilihan yang diambil dari suatu himpunan pilihan yang tersedia.

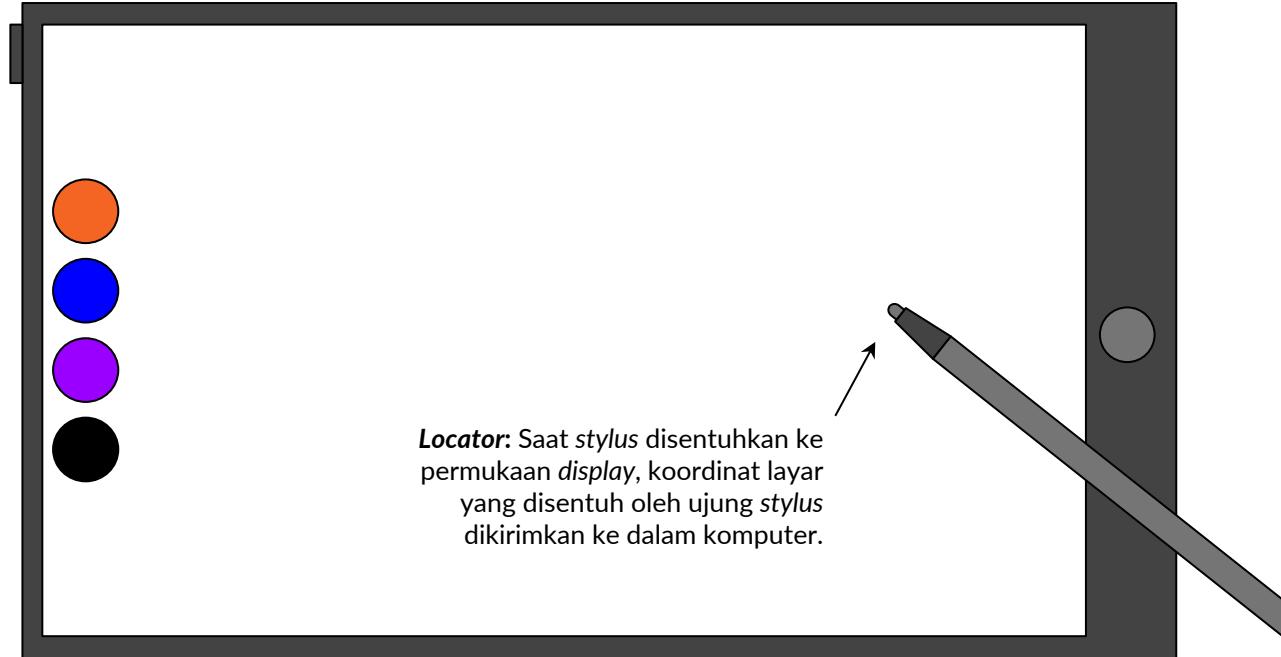
Pick: Identitas objek grafis yang ditampilkan pada perangkat *display* dan sedang dipilih oleh pengguna.

Setiap data memiliki pasangan perangkat masukan logikal yang menjembatani antara masukan pengguna dengan komputer.

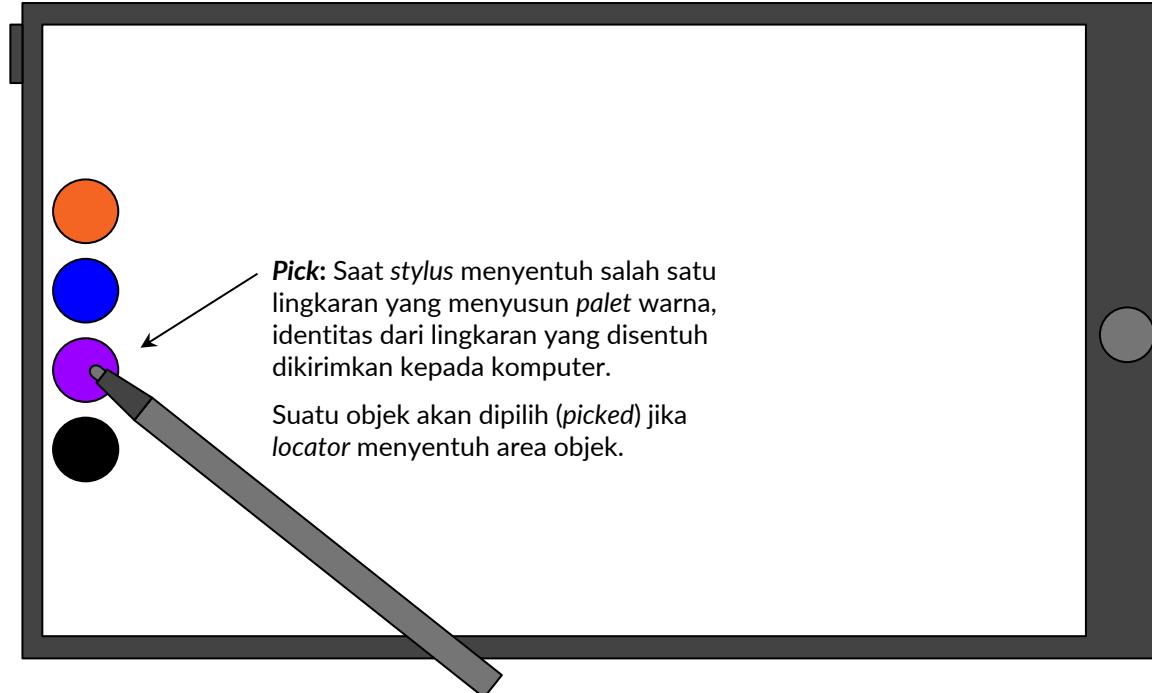


Misalkan kalian menggunakan *pressure-sensitive stylus* untuk menulis di permukaan layar sentuh tablet. Identifikasi apa saja data yang terlibat pada konteks penggunaan berikut.

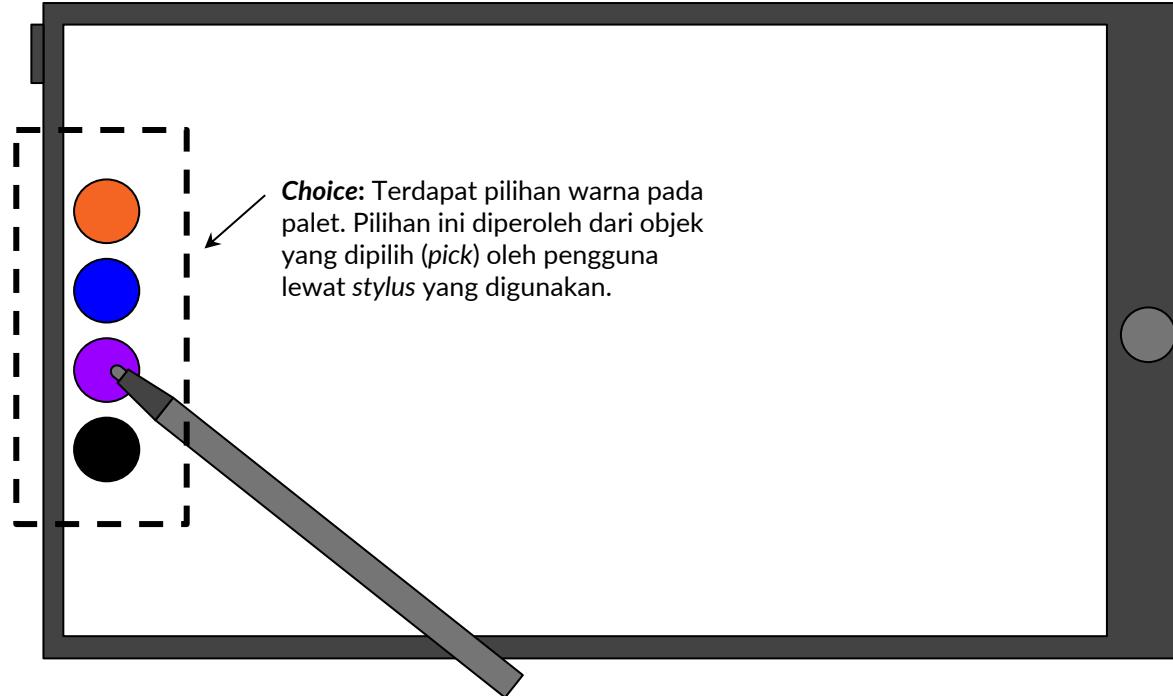
Dari Perangkat Fisikal ke Perangkat Logikal



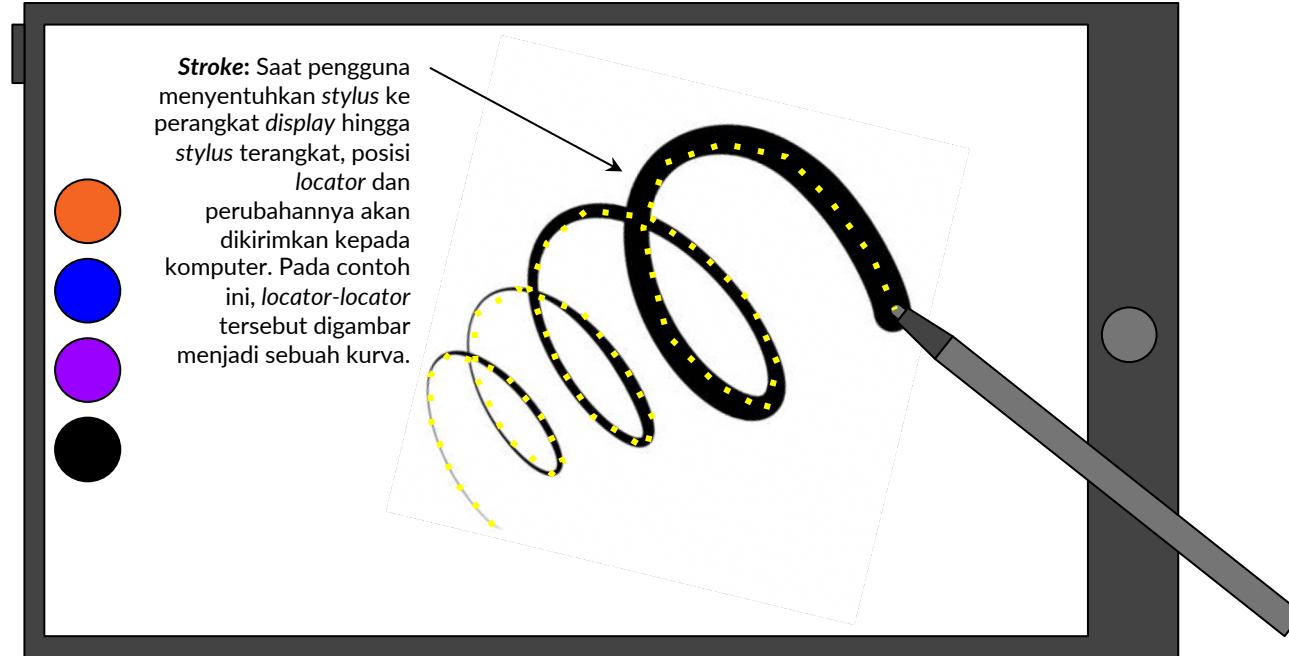
Dari Perangkat Fisikal ke Perangkat Logikal



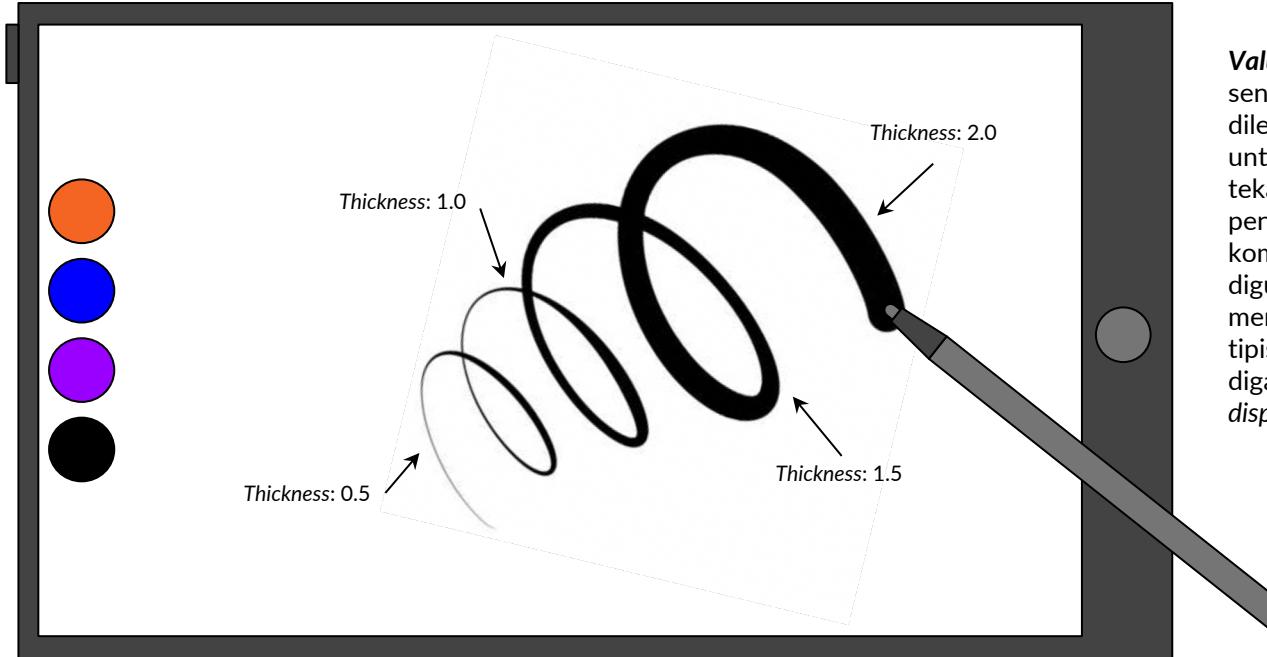
Dari Perangkat Fisikal ke Perangkat Logikal



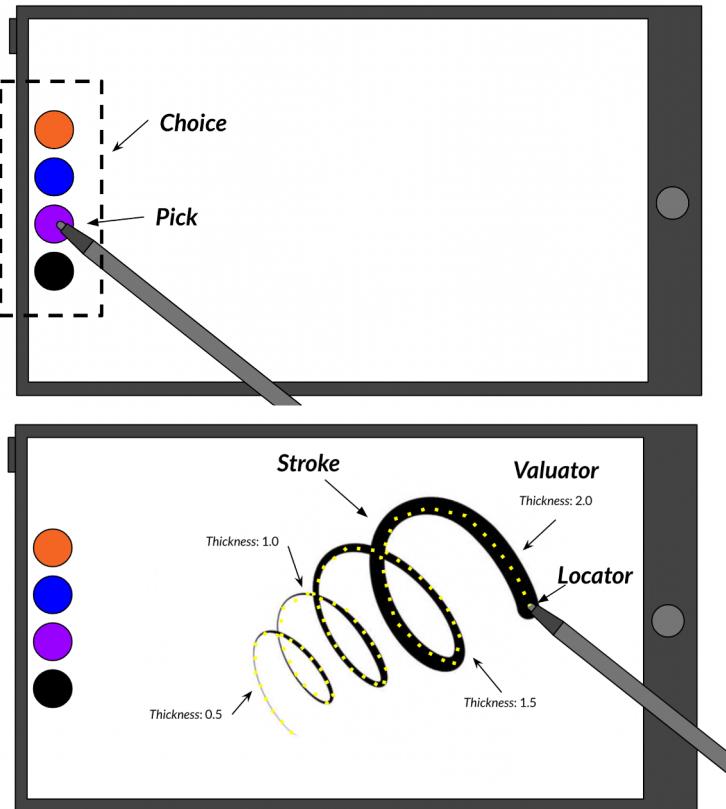
Dari Perangkat Fisikal ke Perangkat Logikal



Dari Perangkat Fisikal ke Perangkat Logikal



Valuator: Pressure sensitive stylus dilengkapi mekanisme untuk mengubah tekanan yang diberikan pengguna pada komputer. Tekanan ini digunakan untuk menentukan tebal tipisnya garis yang digambar di perangkat display.



Dari Perangkat Fisikal ke Perangkat Logikal

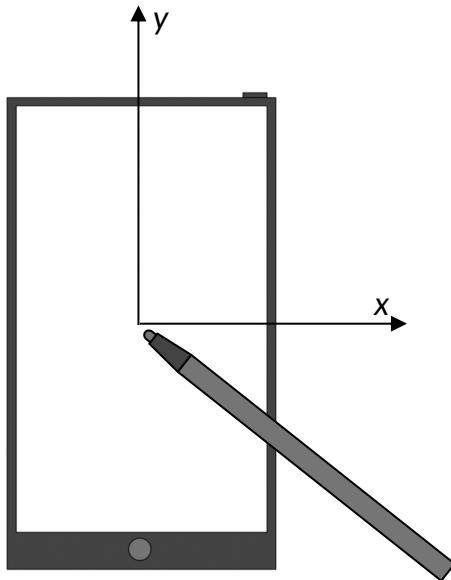
Ilustrasi tersebut memperlihatkan bagaimana pengguna menggunakan perangkat masukan fisikal untuk menghasilkan data yang dikirimkan ke komputer.

Saat menggunakan perangkat fisikal untuk melakukan sesuatu, pengguna dapat menghasilkan lebih dari satu data.

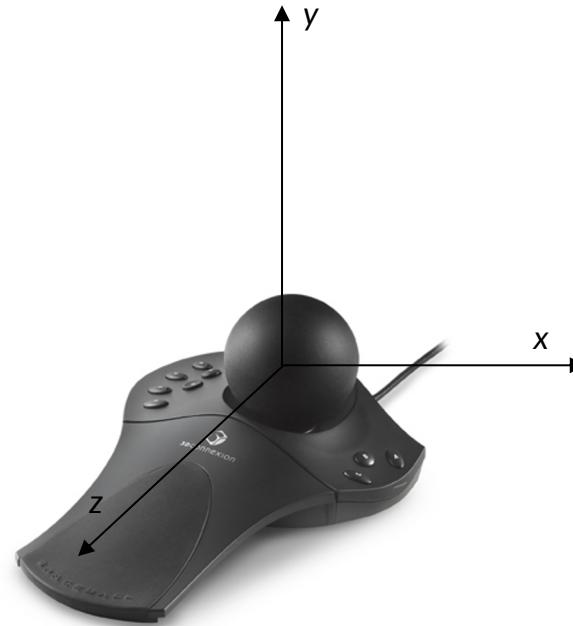
Perangkat dapat berbeda, namun prinsip-prinsip ini tetap berlaku secara standar.

Data berupa *string* lazimnya dikirim menggunakan perangkat seperti *keyboard*.

Dua Dimensi vs Tiga Dimensi



Locator pada tablet.



Locator pada SpaceBall.



Locator pada perangkat virtual reality.



IPB University
— Bogor Indonesia —

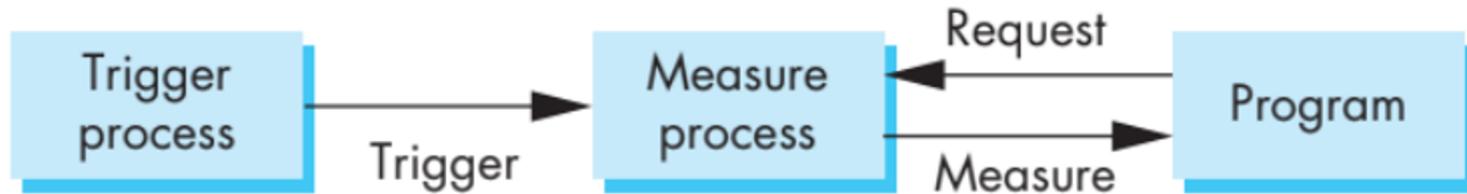
Departemen
Ilmu Komputer

Bagian 2: *Input Mode*

Trigger dan Measure

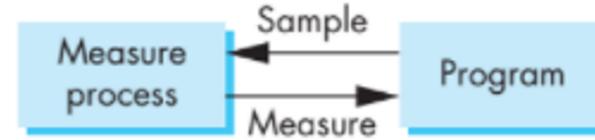
- Perangkat *input* mengirimkan *input* terkait dengan dua hal, yaitu *measure* dan *trigger*.
- *Trigger* menunjukkan saat pengguna mengirimkan sinyal ke komputer.
- *Measure* adalah informasi yang dikirimkan ke komputer.
- Contoh: pada saat mengetik, *trigger* adalah ketika pengguna menekan tombol pada *keyboard*, dan *measure* yang dikirimkan adalah kode ASCII dari tombol yang diketik.
- Ada beberapa *input mode* yang mengatur mekanisme pengiriman *trigger* dan *measure* ini: *request mode*, *sample mode*, dan *event mode*.

Request Mode



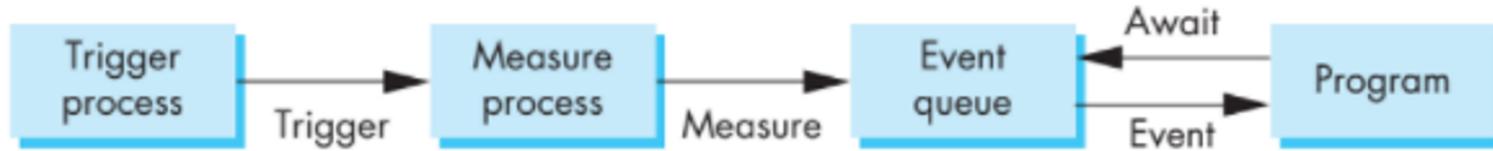
- Measure hanya akan dikirimkan kepada program apabila program mengirimkan request.
- Misalnya, menggunakan perintah scanf pada bahasa C untuk menerima entri data berupa teks dari pengguna.

Sample Mode



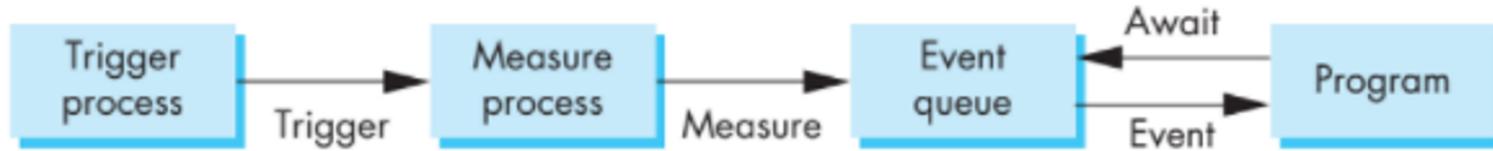
- *Measure* terus-menerus dikirimkan kepada program sehingga trigger tidak diperlukan.
- Contoh: penggunaan *mouse* pada sistem operasi.

Event Mode



- Berguna jika sebuah aplikasi menggunakan banyak perangkat *input* sekaligus (mouse, keyboard, joystick, dll).
- Menggunakan *event queue* untuk menampung seluruh kejadian yang terjadi, kemudian *event queue* dikerjakan satu per satu.
- Memperhatikan *event* dan *callback function* untuk mendeteksinya terjadinya suatu *event* pada aplikasi.

Event Mode: Contoh Event



- *Mouse Event*: pergerakan kursor, menekan tombol, menekan serangkaian tombol, memutar *mouse wheel*.
- *Windows Event*: membuka dan menutup, mengubah ukuran.
- *Keyboard Event*: menekan dan melepas tombol, menekan lebih dari satu tombol sekaligus, menahan tombol.
- dll...

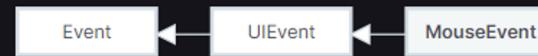
Pelajari Dokumentasi untuk *Event* yang Didukung

MouseEvent

The `MouseEvent` interface represents events that occur due to the user interacting with a pointing device (such as a mouse). Common events using this interface include `click`, `dblclick`, `mouseup`, `mousedown`.

`MouseEvent` derives from `UIEvent`, which in turn derives from `Event`. Though the `MouseEvent.initMouseEvent()` method is kept for backward compatibility, creating of a `MouseEvent` object should be done using the `MouseEvent()` constructor.

Several more specific events are based on `MouseEvent`, including `WheelEvent`, `DragEvent`, and `PointerEvent`.



<https://developer.mozilla.org/en-US/docs/Web/API/MouseEvent>

[https://www.youtube.com/
watch?v=vPsFPmgToYM](https://www.youtube.com/watch?v=vPsFPmgToYM)

Penutup

Curiosity and Possibility



Ivan Sutherland

2016 Inductee - Sketchpad — A Man-Machine
Graphical Communication System



Terima Kasih

**Auzi Asfarian, M Asyhar Agmalaro,
Endang P Giri, Shervie N Neyman**

Departemen Ilmu Komputer, FMIPA
IPB University (Institut Pertanian Bogor), Bogor, Indonesia
asfarian@apps.ipb.ac.id | ux@apps.ipb.ac.id