COSE436

Lecture 8: 3D Data Acquisition & Representation

Oct. 15, 2024

Won-Ki Jeong
(wkjeong@korea.ac.kr)

KOREA UNIVERSITY

Outline

- 3D data acquisition
- Surface representations

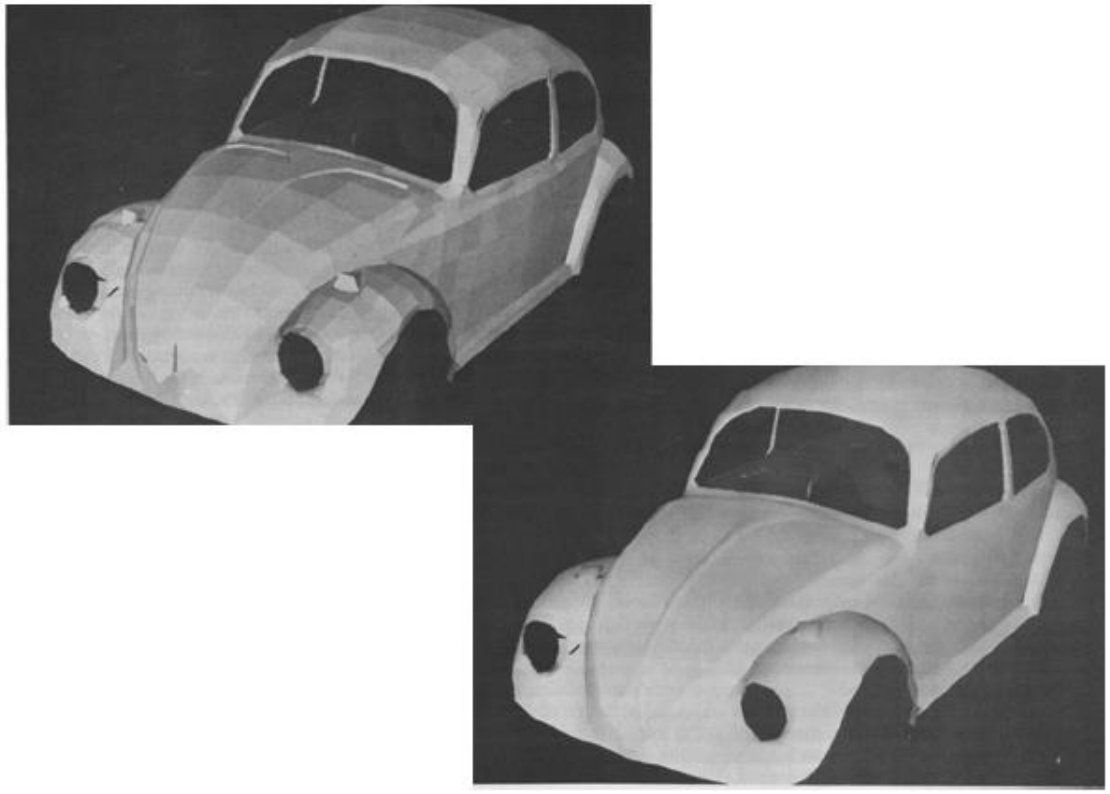
Outline

- 3D data acquisition
- Surface representations

3D Data Acquisition

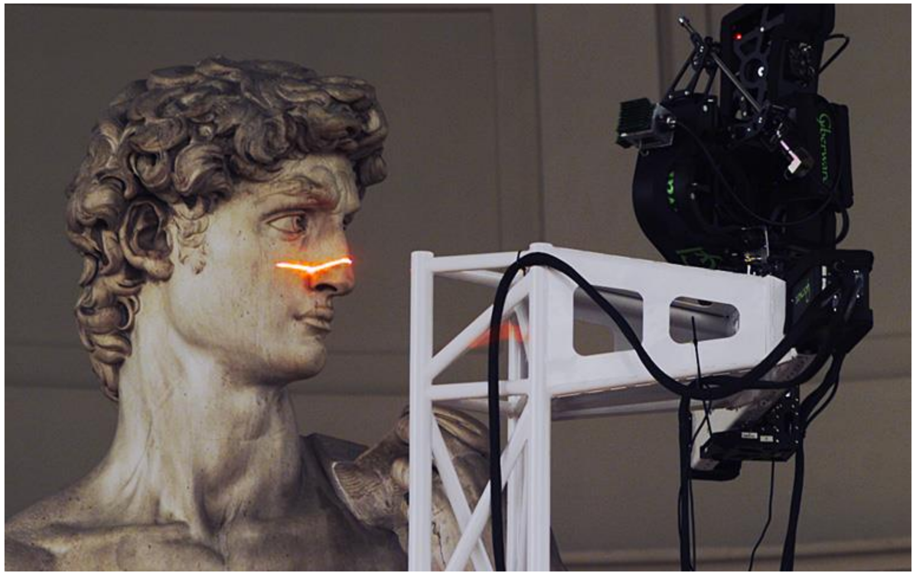
- Digitize 3D points on the objects

Digitizing Ivan Sutherland's VW bug


Utah VW bug 3D model

Carson

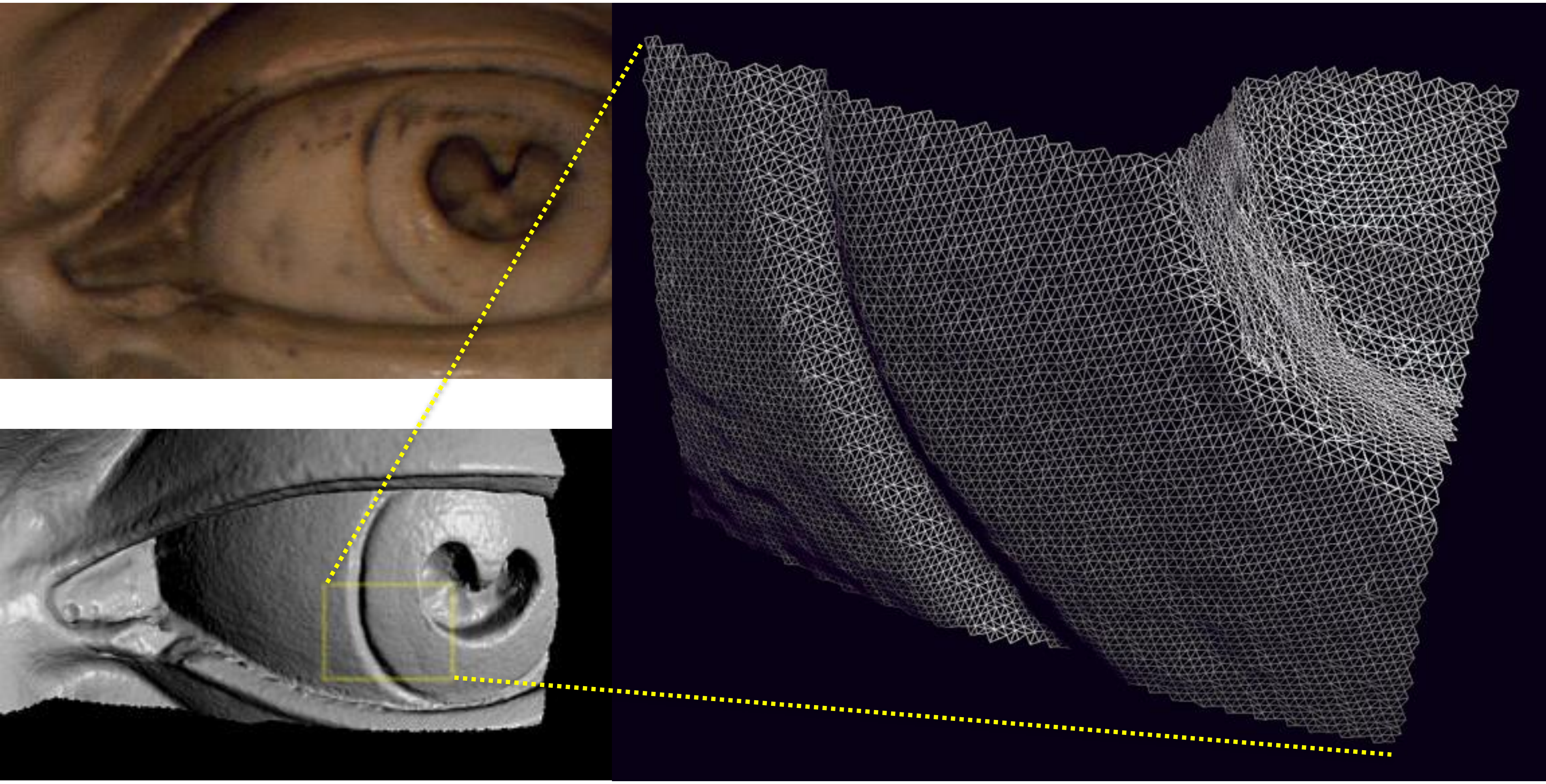
The Digital Michelangelo Project



Stanford University and the University of Washington spent 1998-99
http://graphics.stanford.edu/projects/mich/color-david/david-both-bal-ssh.jpg
0.25mm / pixel scan of 5m David statue, 940 million polygons, the largest scanned 3D data

Levoy

Range Data Acquisition



Levoy

Range Acquisition Methods

- Contact
  - Touch probe
- Transmissive
  - CT, MRI, Ultrasound
- Reflective
  - Radar, sonar, optical
- Passive
  - Shape from shading
- Active
  - Time of flight, trianglulation

Touch Probes

- Contact method

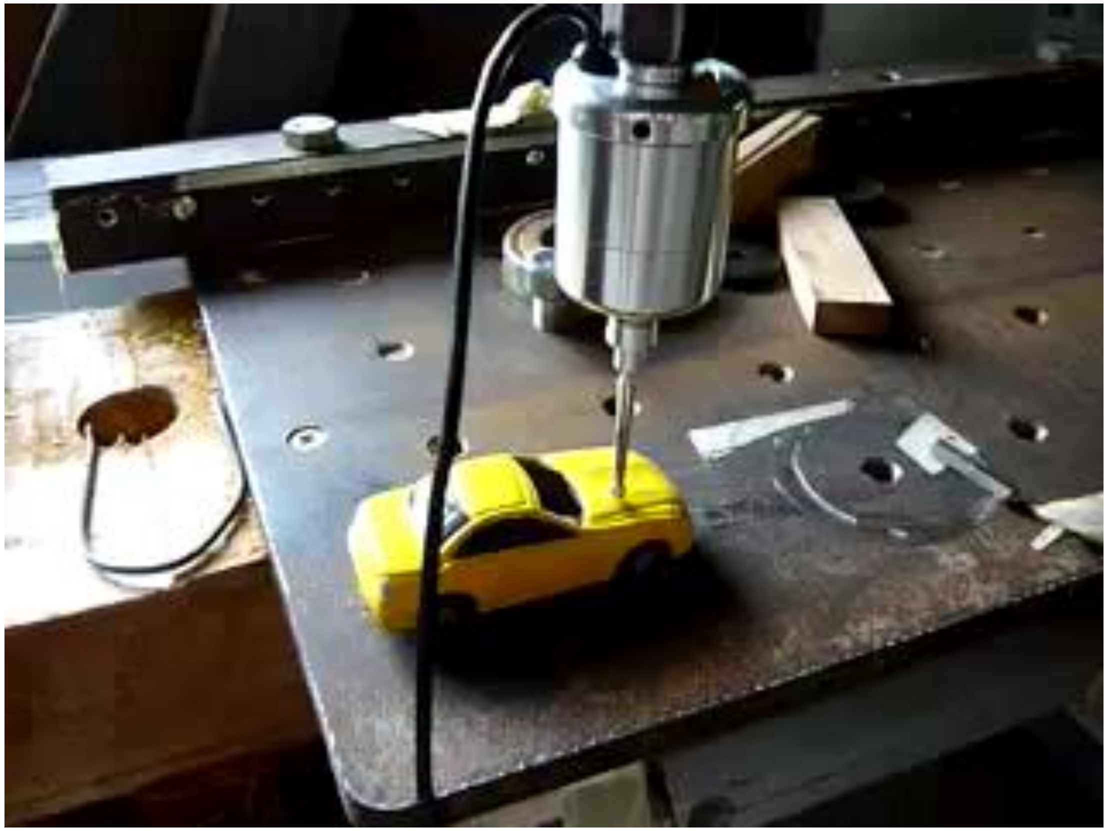
Faro arm

CNC step

http://www.youtube.com/watch?v=R9i4DbwZrKI&feature=related

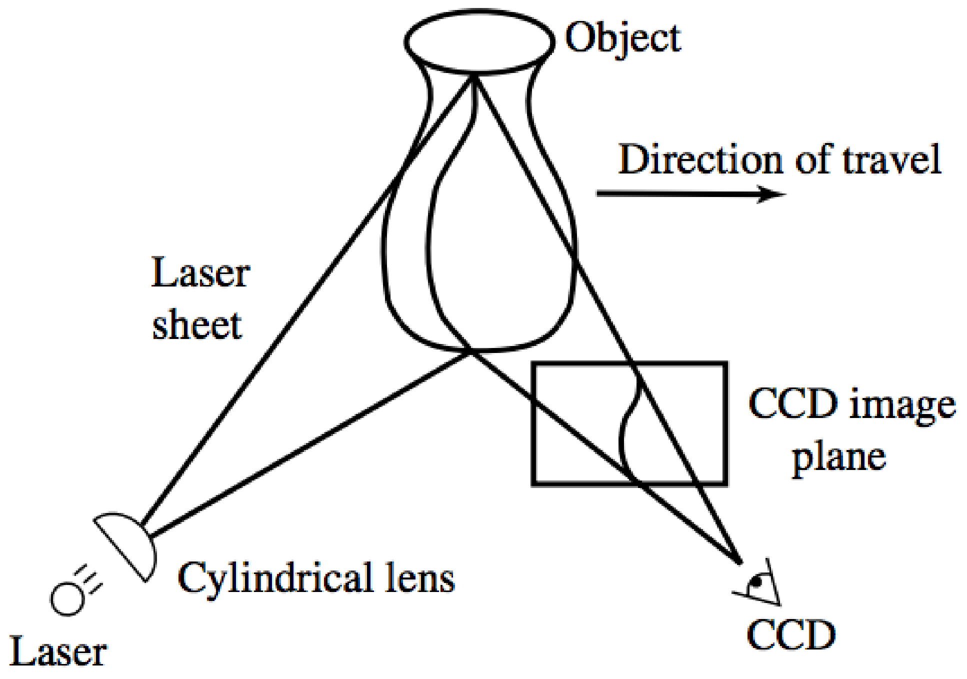
Touch Probes

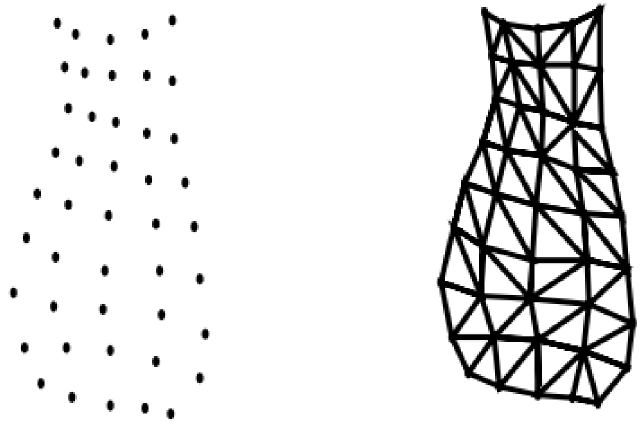
- Contact method



Range Scanners

- Measure distances to the object
  - Convert to 3D location



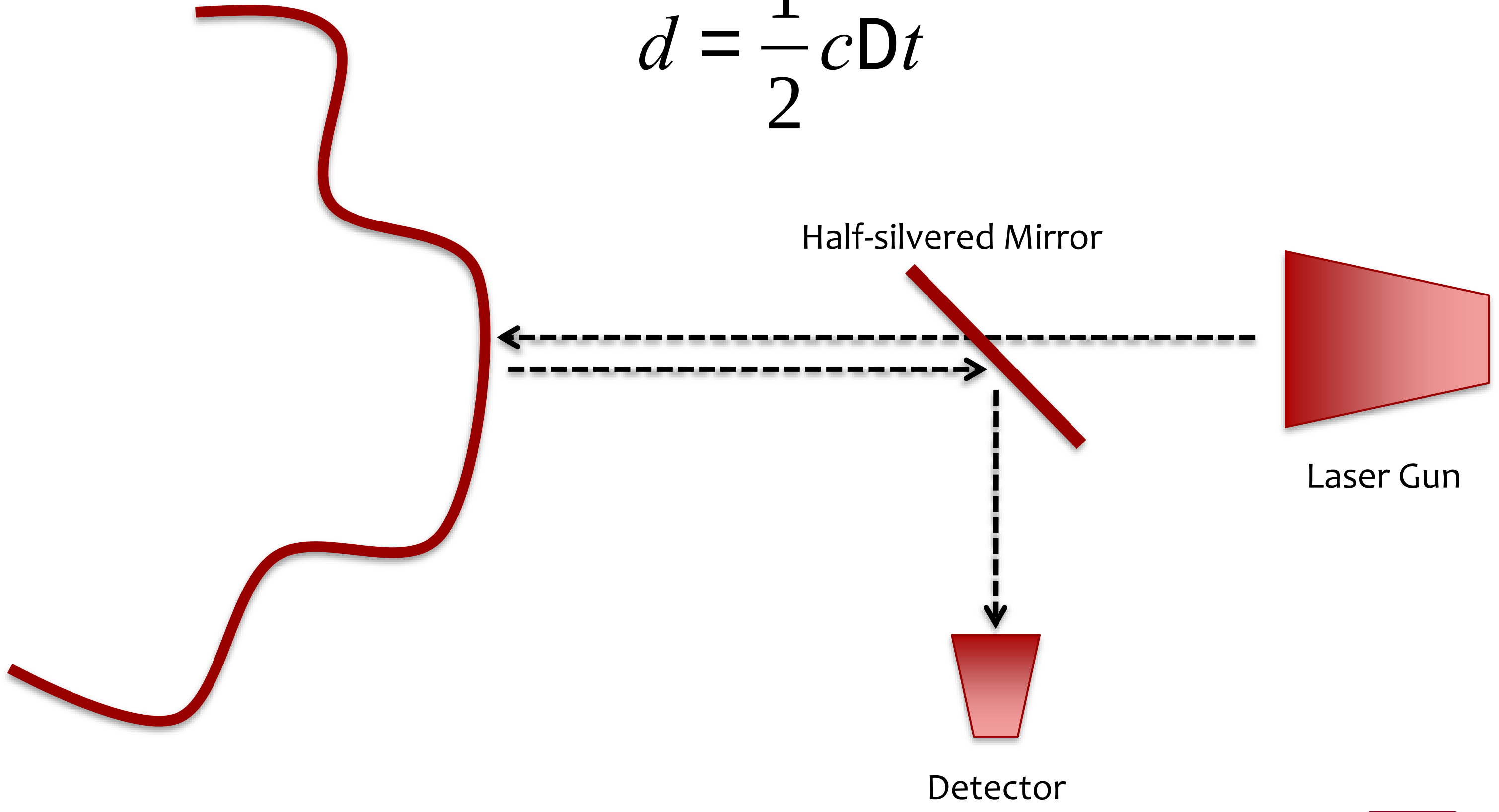


Curless et al.

Time of Flight (LIDAR)

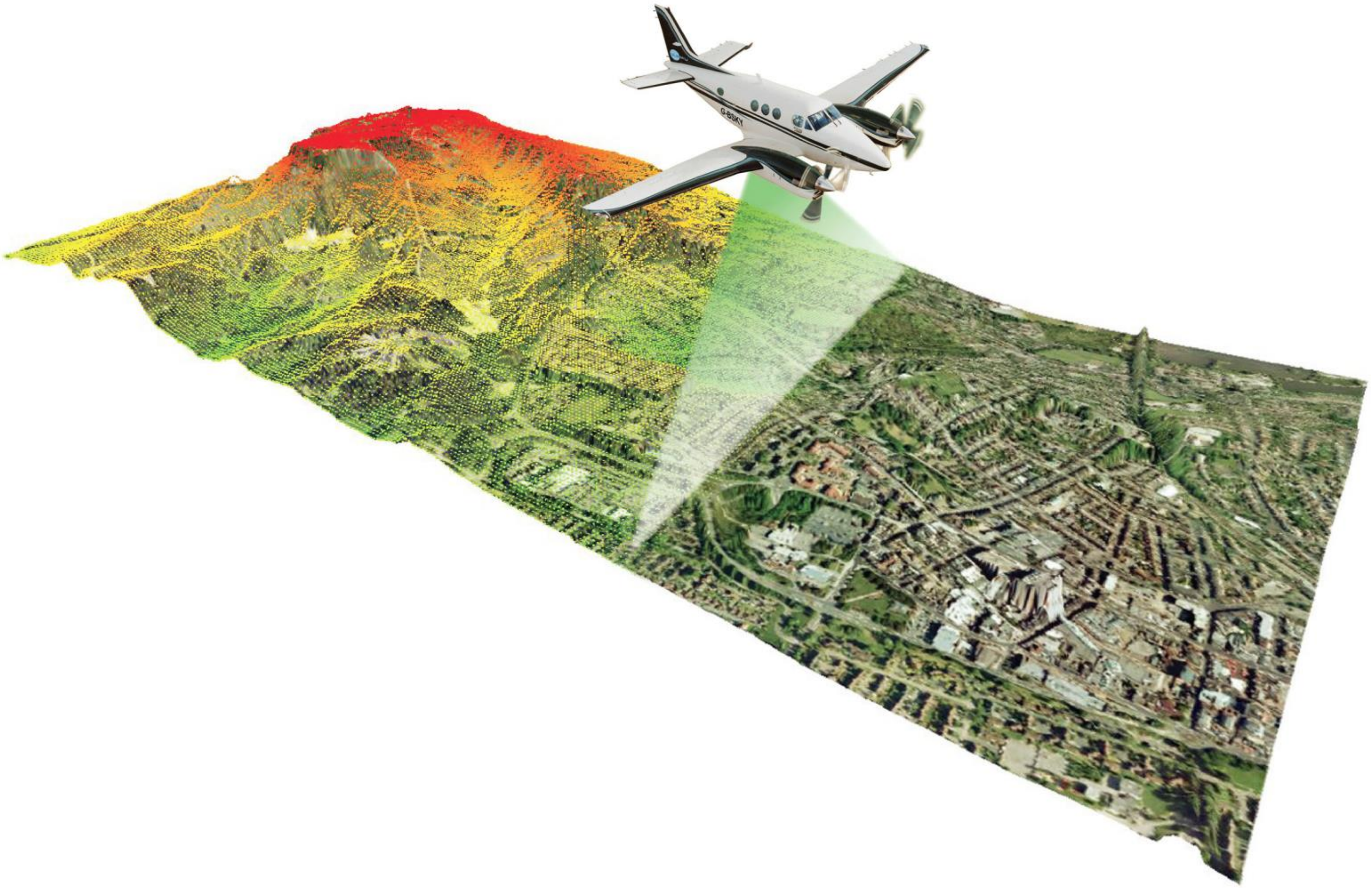
- Measure laser travel time (*c*: speed of light)

$$d = \frac{1}{2} c \Delta t$$

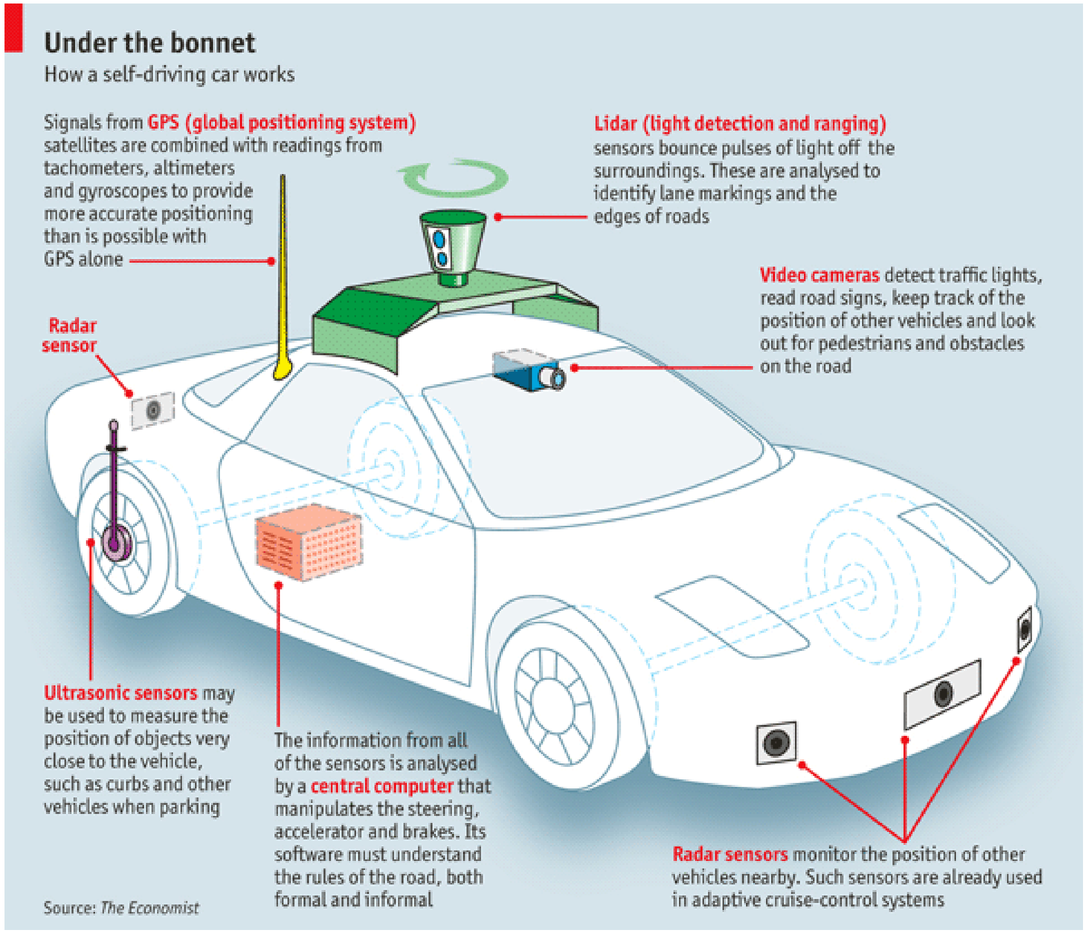


LIDAR

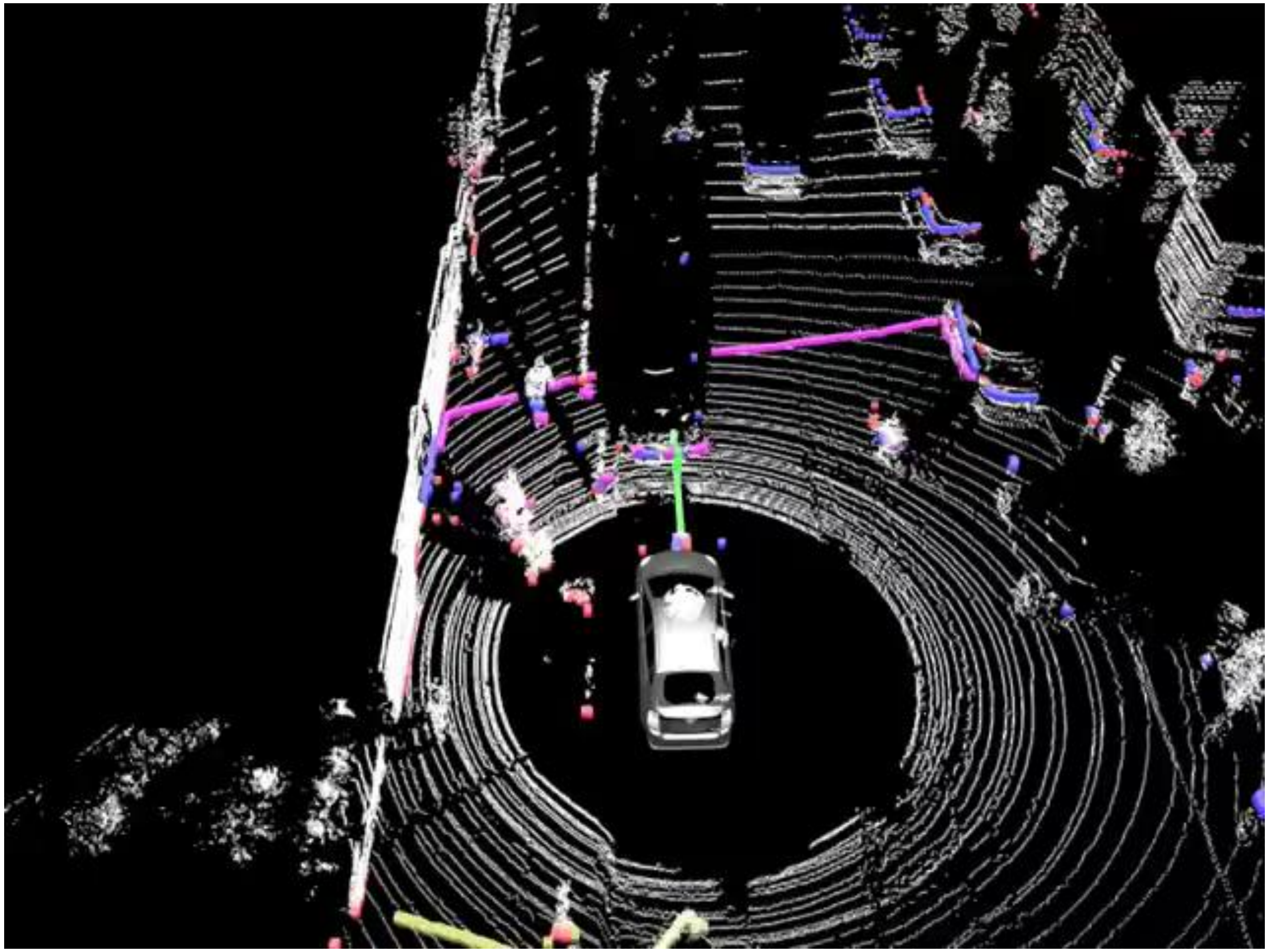
LIDAR



LIDAR


Under the bonnet
How a self-driving car works
Signals from GPS (global positioning system) satellites are combined with readings from tachometers, altimeters and gyroscopes to provide more accurate positioning than is possible with GPS alone
Lidar (light detection and ranging) sensors bounce pulses of light off the surroundings. These are analysed to identify lane markings and the edges of roads
Video cameras detect traffic lights, read road signs, keep track of the position of other vehicles and look out for pedestrians and obstacles on the road
Radar sensor
Ultrasonic sensors may be used to measure the position of objects very close to the vehicle, such as curbs and other vehicles when parking
The information from all of the sensors is analysed by a central computer that manipulates the steering, accelerator and brakes. Its software must understand the rules of the road, both formal and informal
Radar sensors monitor the position of other vehicles nearby. Such sensors are already used in adaptive cruise-control systems
Source: The Economist

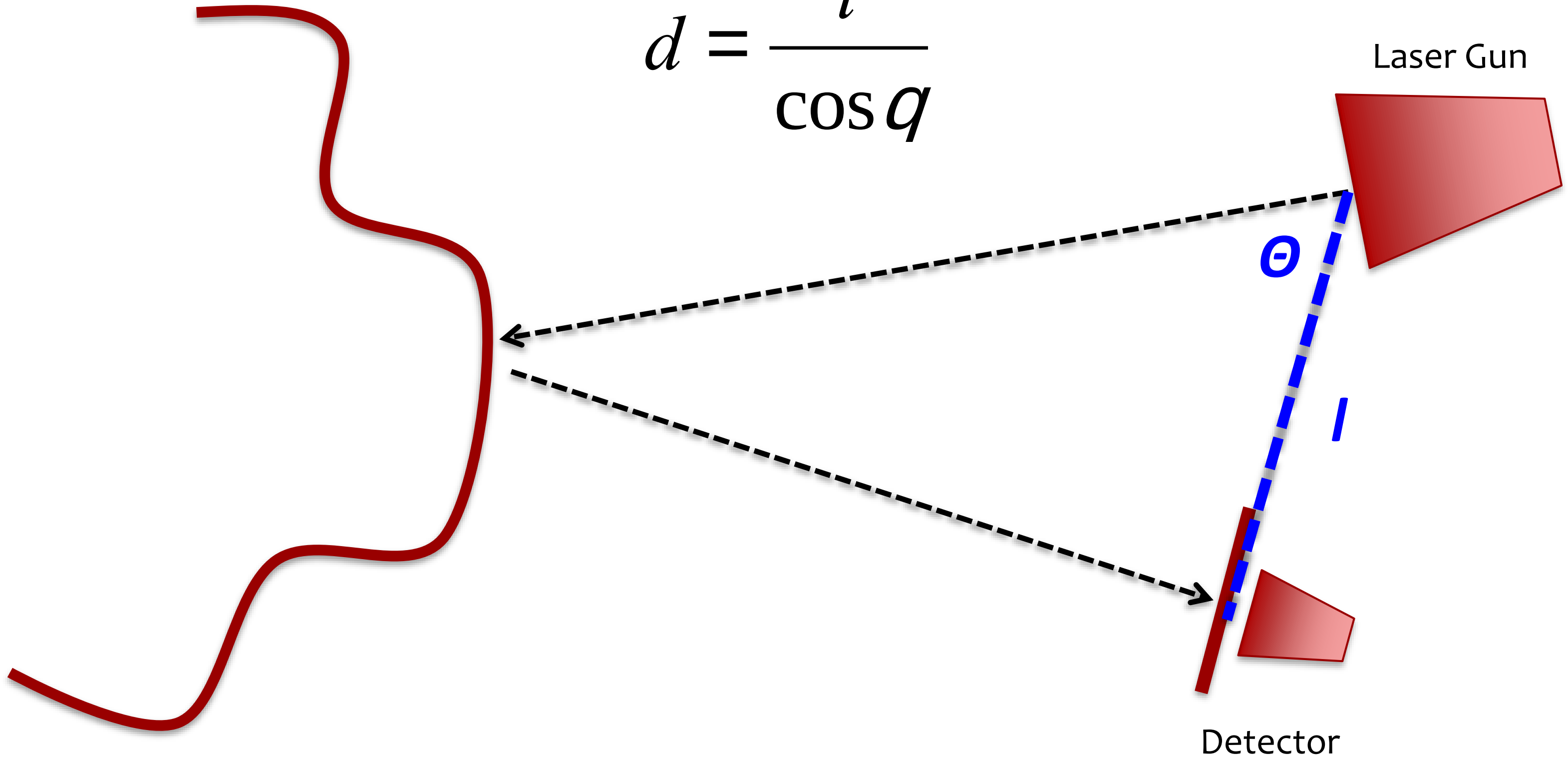
LIDAR Visualization



Triangulation

- Geometric principle

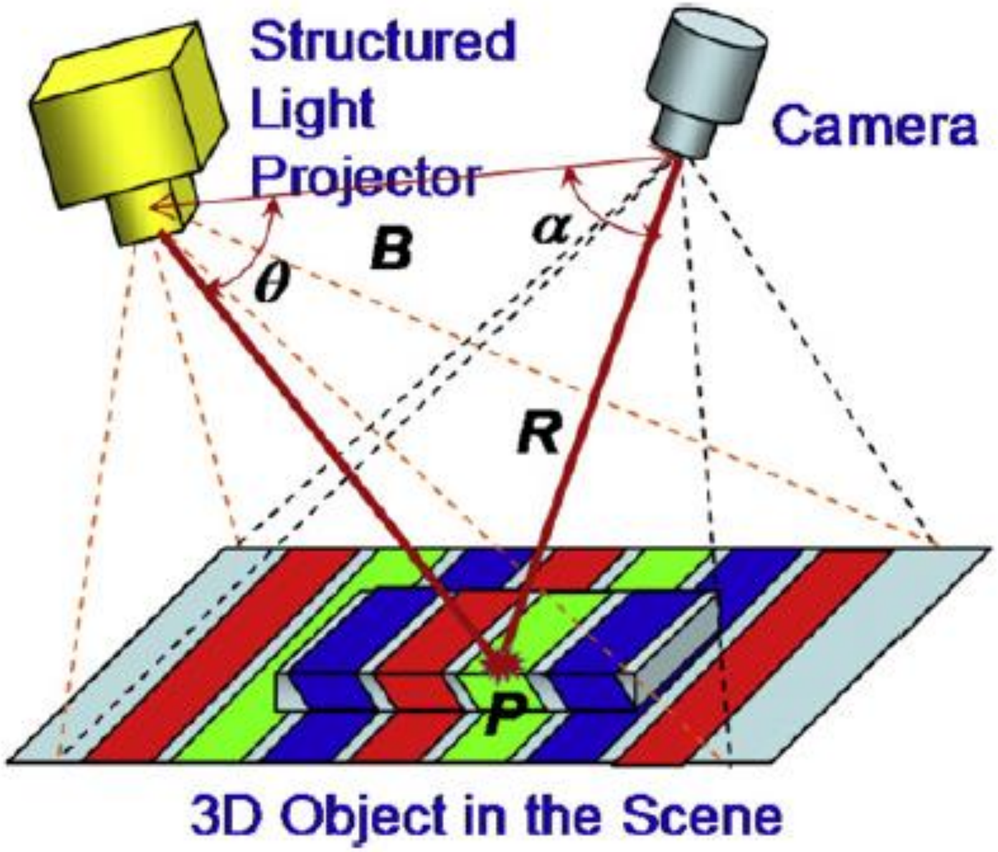
$$d = \frac{l}{\cos q}$$



Triangulation, Single Scan Line

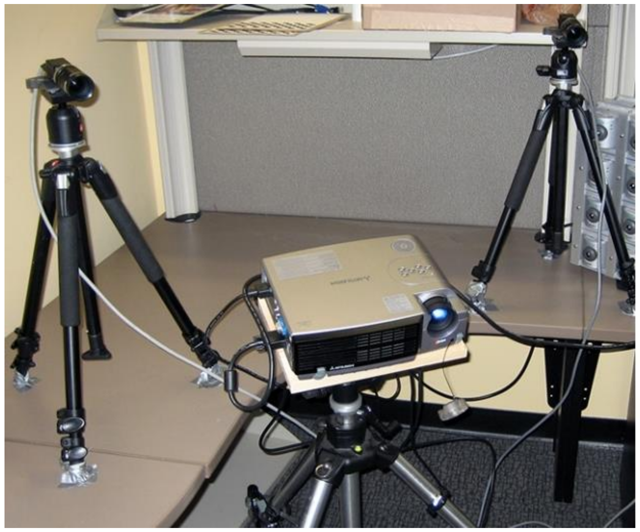
Structured Light

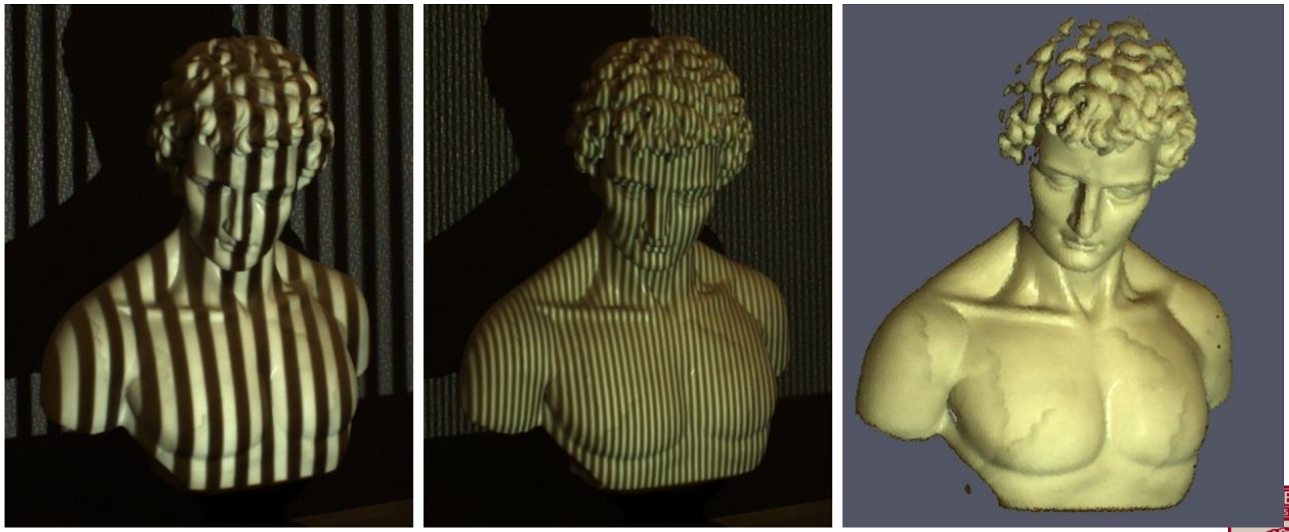
- Image-based technique, multiple scanning lines



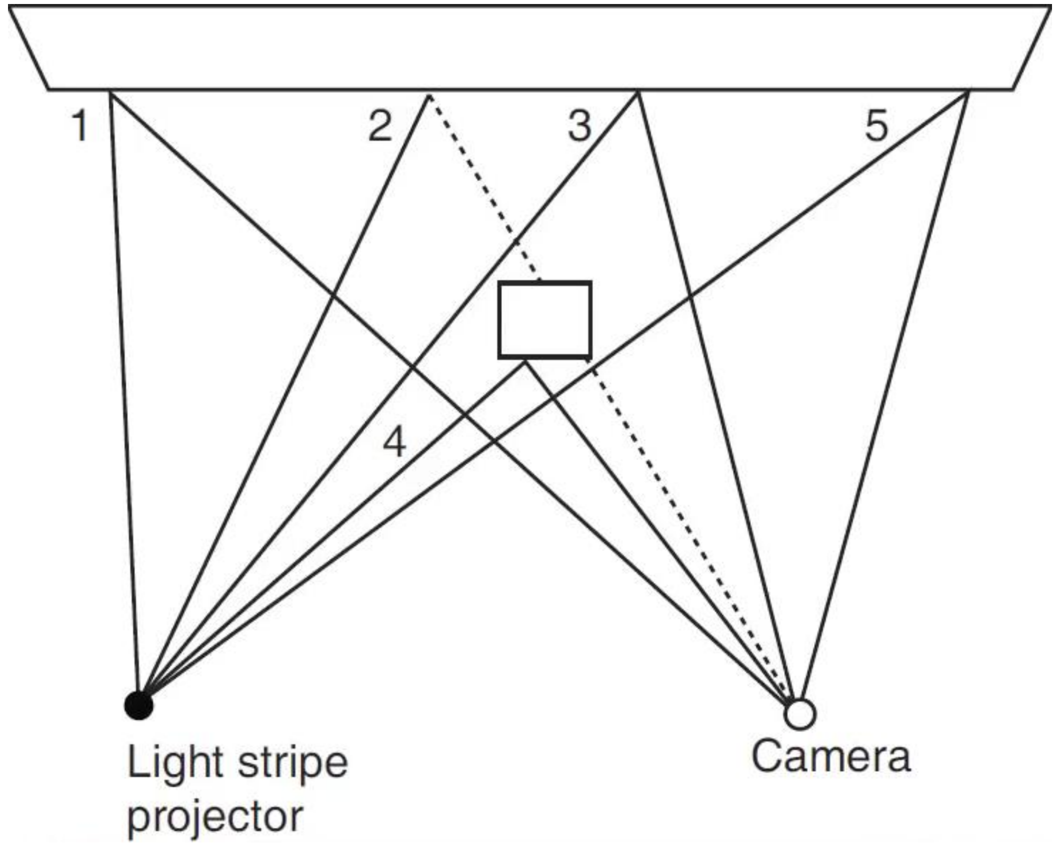
Geng

Structured Light



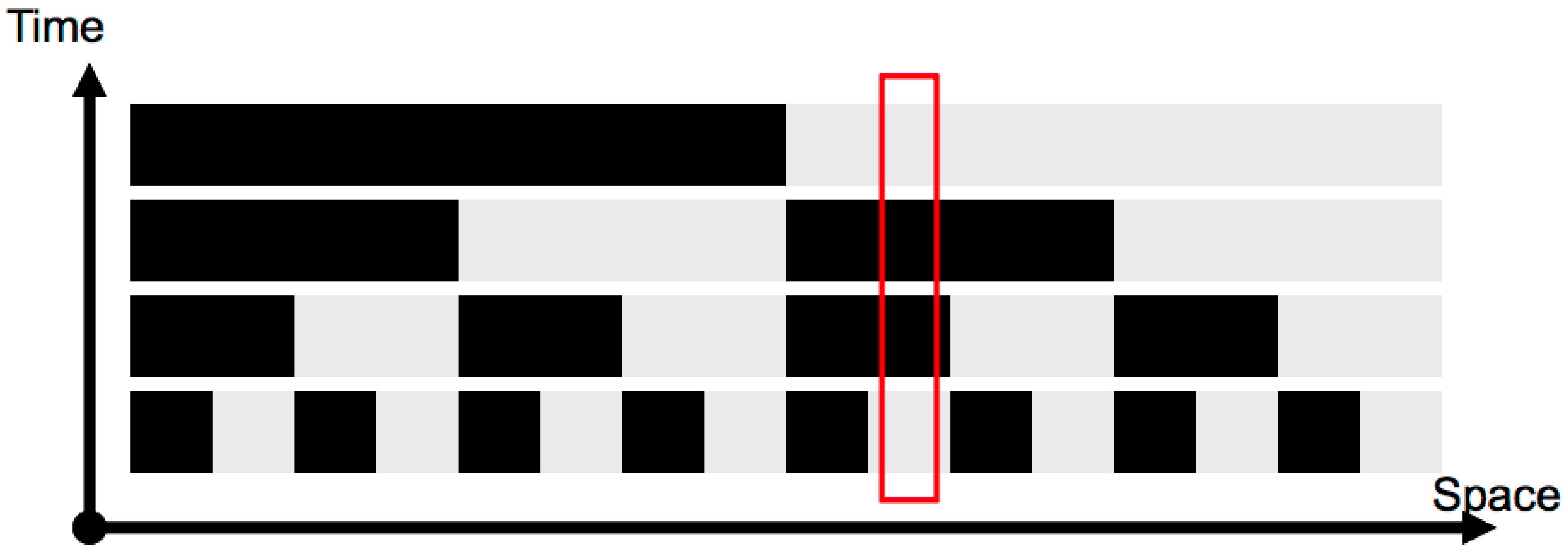


Structured Light Problem



Structured Light Encoding

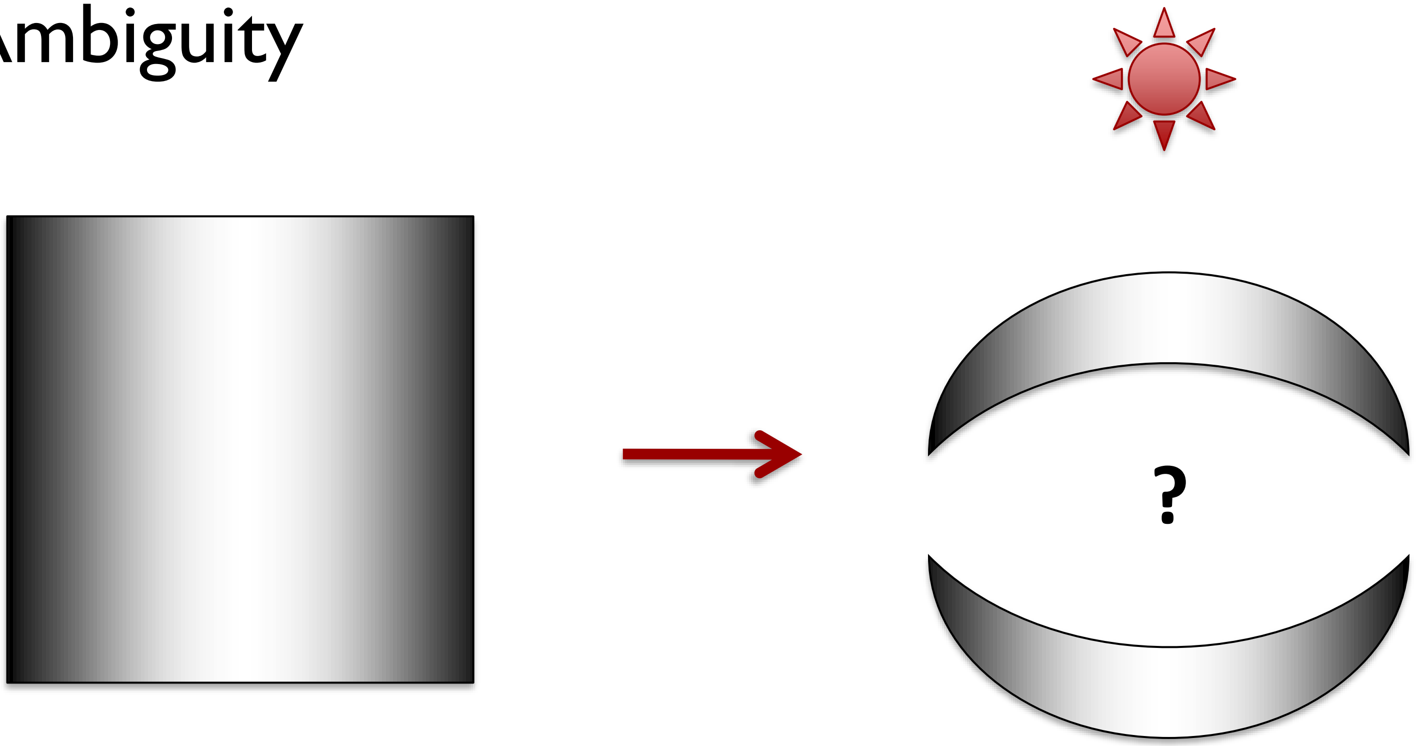
- Binary coding



1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Shape from Shading

- Known reflectance and light source
- Estimate relative surface normal
- Ambiguity



Problems in Range Scanning

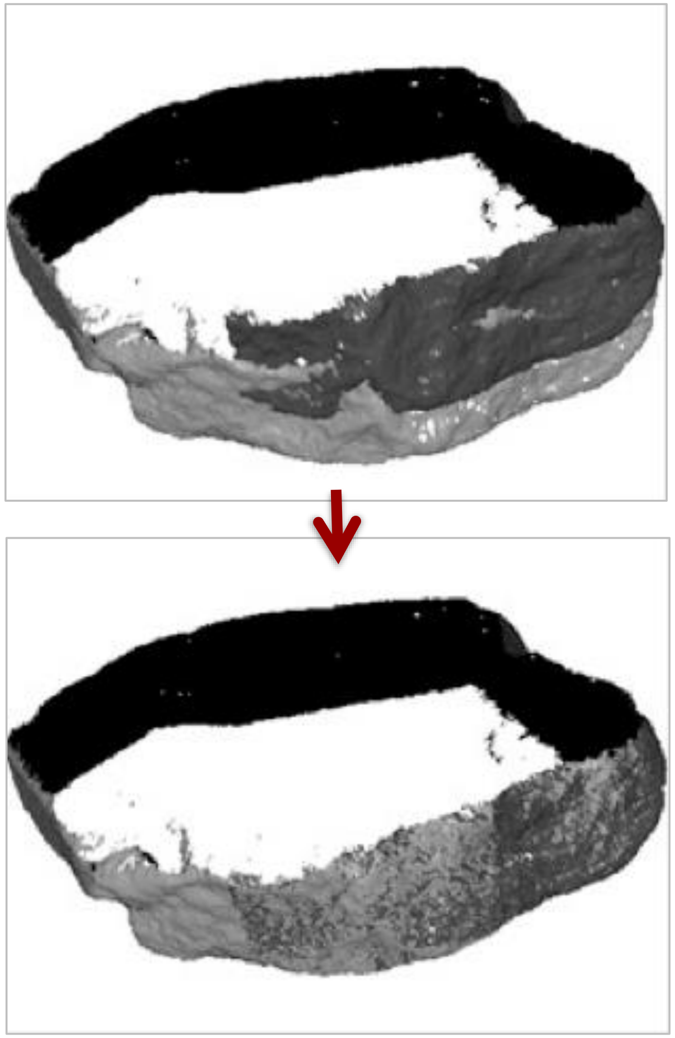
- Multiple patches
  - Limited field-of-view
- Artifacts
  - Holes, noise, topological errors
- Post-processing is required!

Range Processing Pipeline

- Manual initial alignment
- ICP alignment between patches
- Merging using volumetric method

Range Processing Pipeline

- Manual initial alignment
- ICP alignment between patches
- Merging using volumetric method

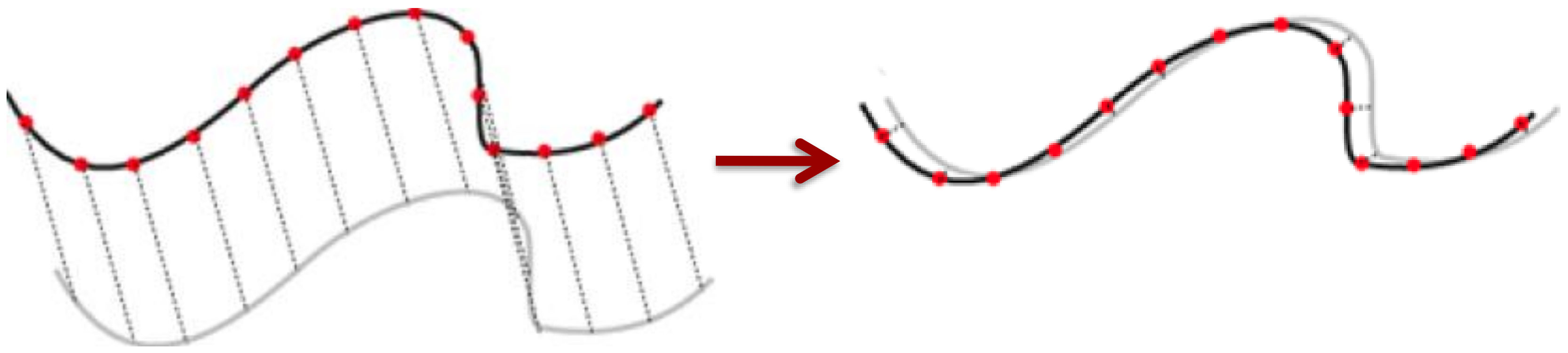


rusinkiewicz

Iterative Closest Point (ICP)

- Algorithm

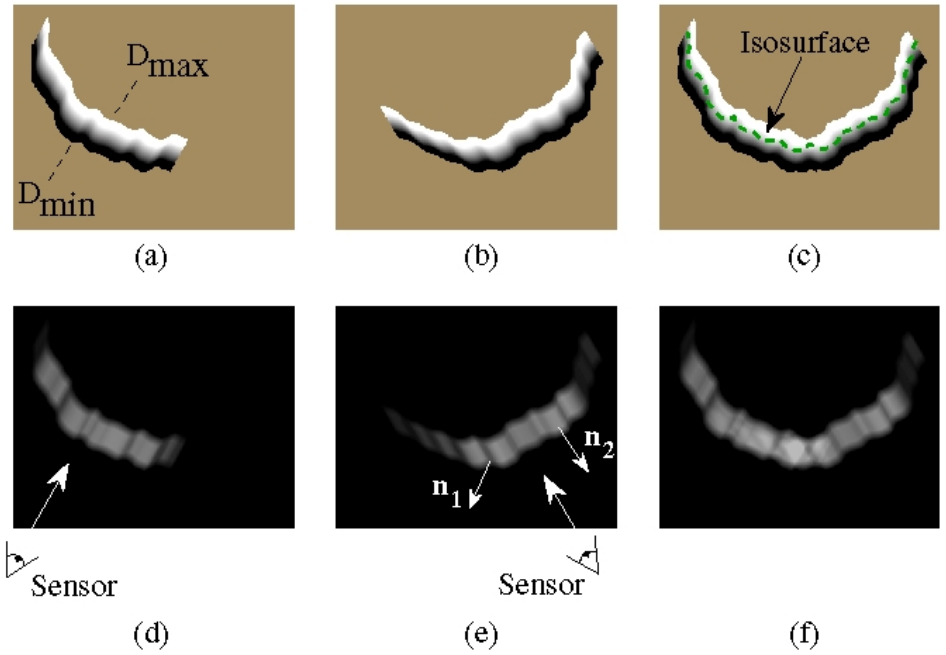
```
1. Find nearest match per point
2. Estimate transformation parameters
3. Transform the points
4. Iterate above steps until converges
```



rusinkiewicz

Range Processing Pipeline

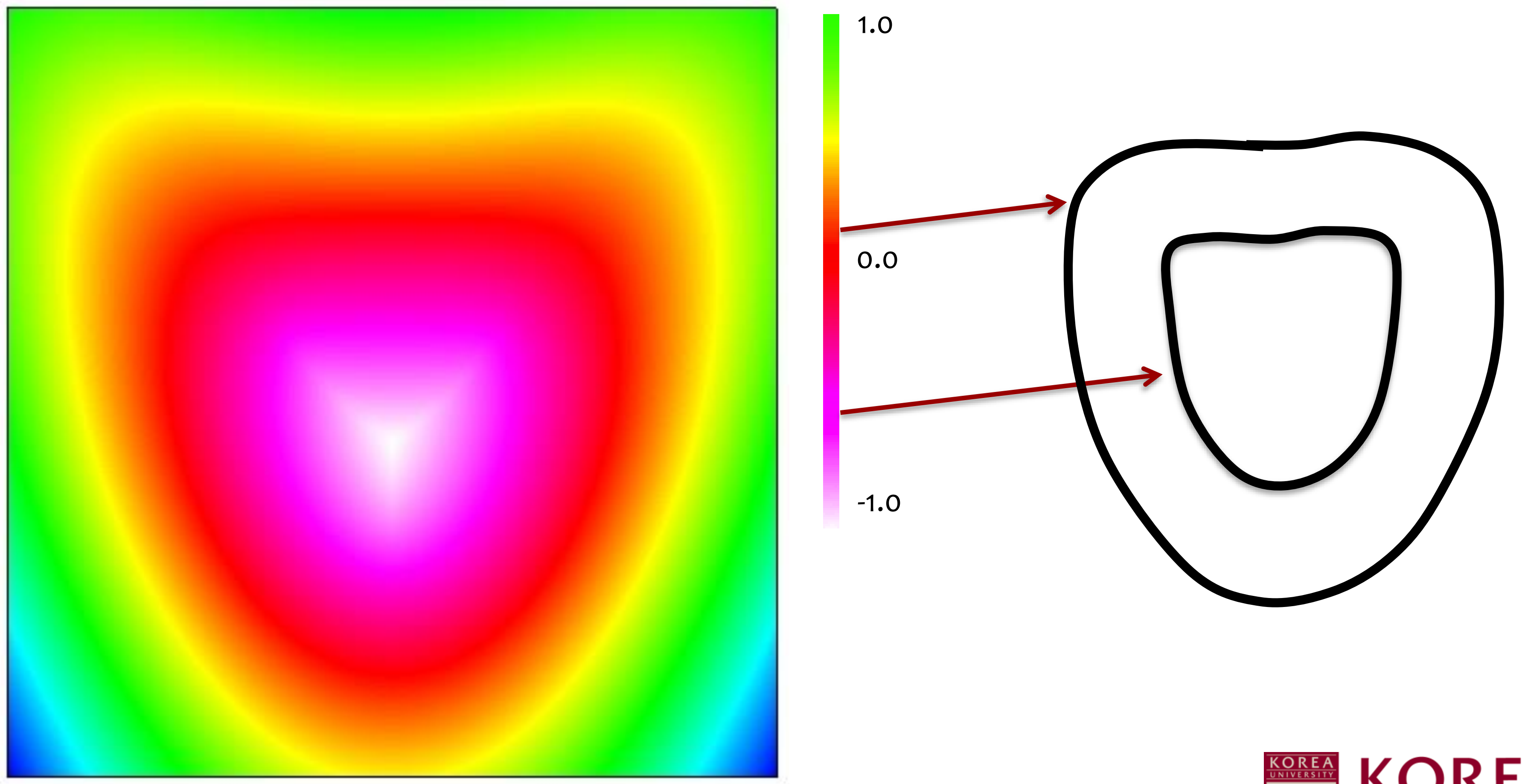
- Manual initial alignment
- ICP alignment between patches
- Merging using volumetric method



Curless & Levoy

Isosurface

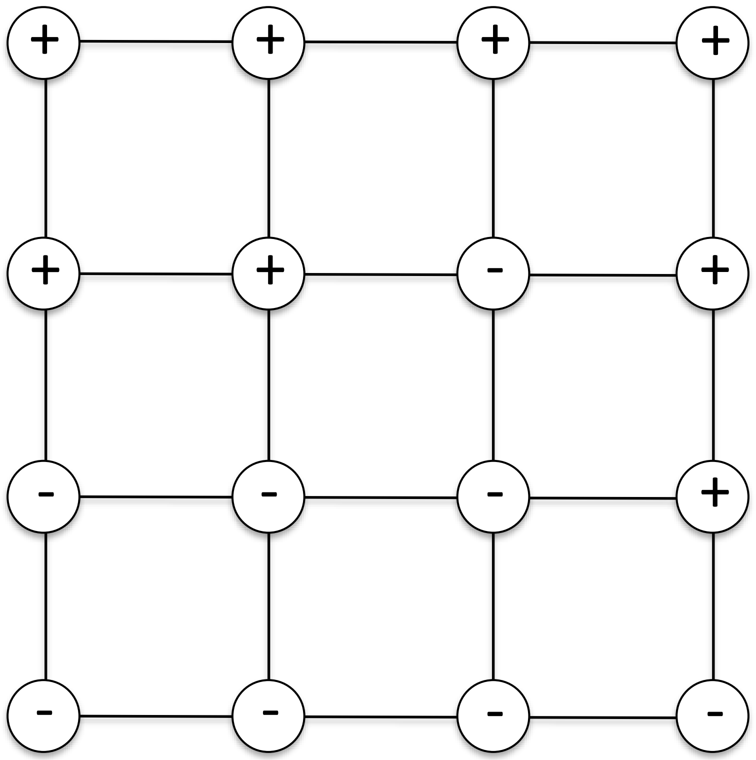
- Signed distance field



CSC

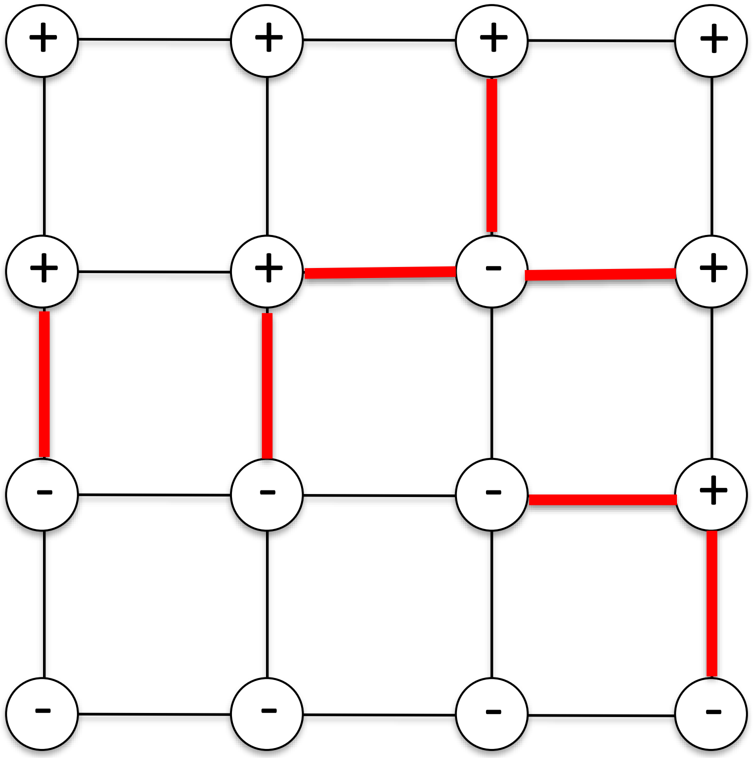
Isosurface

- Find zero-crossing surface



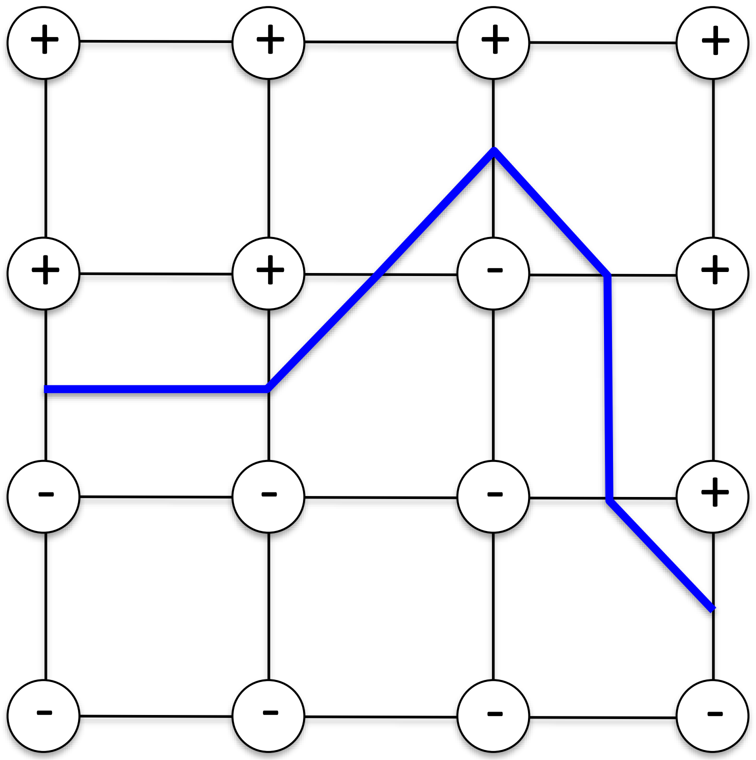
Isosurface

- Find zero-crossing surface



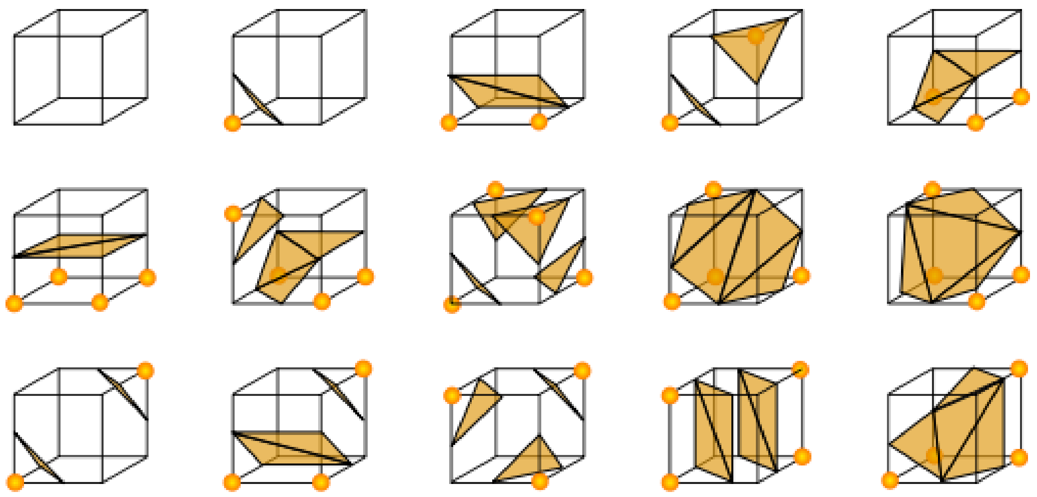
Isosurface

- Find zero-crossing surface



Marching Cubes

- 3D isosurface algorithm
- 2^8 = 256 different topologies
  - Many redundant cases (symmetry)
  - Reduced to only 15 unique cases

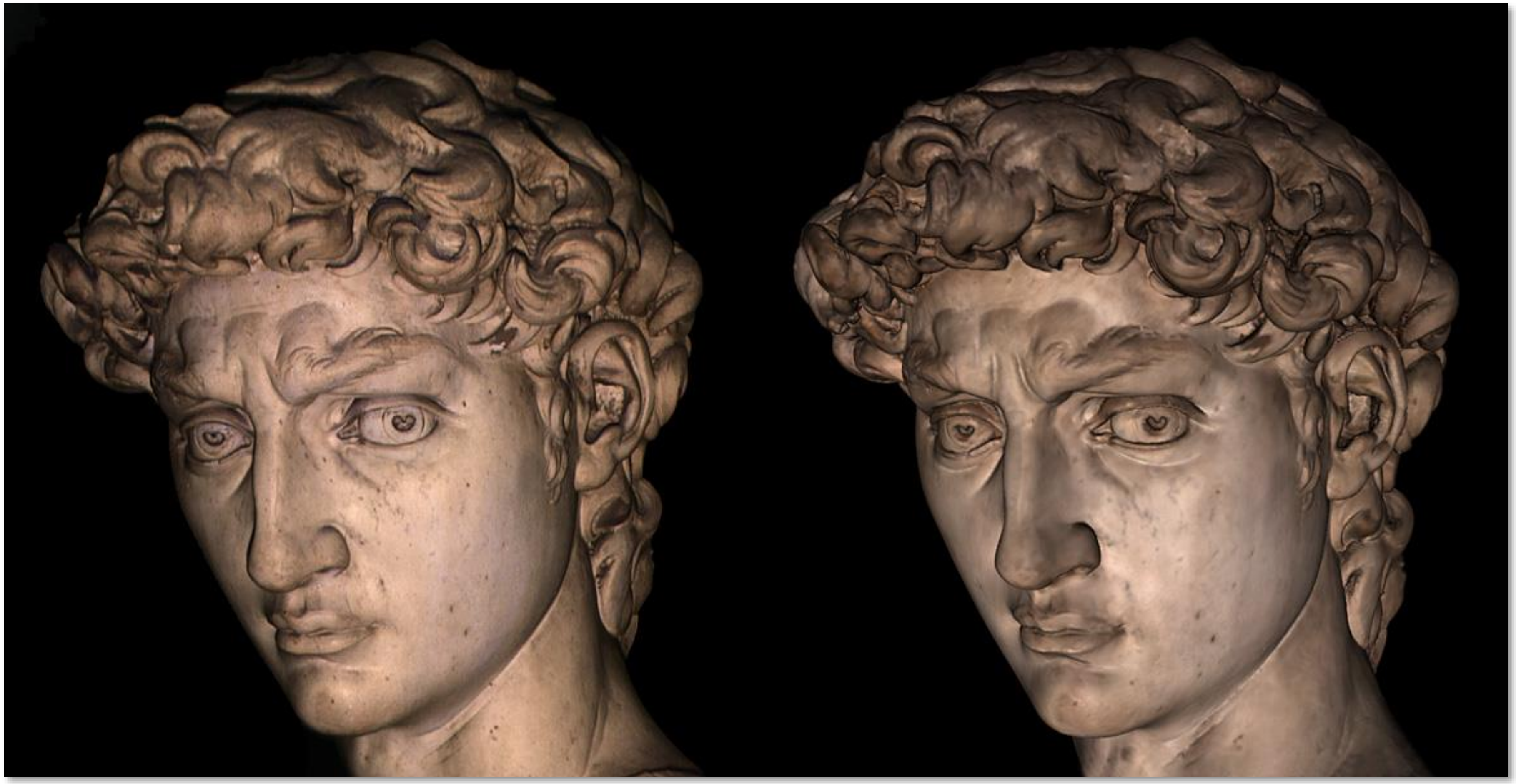


Statistics about the Scan of David

- 480 individual scans
- 0.3 mm spacing
- 2 billion polygons
- 7,000 color images
- 32 Gigabytes
- 22 people @ 30 nights

Rusinkiewicz & Levoy

David Head



Photograph

Rendered

Levoy

Outline

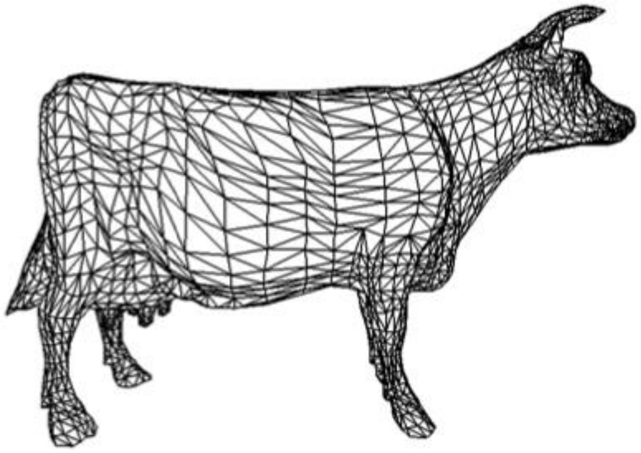
- 3D data acquisition
- Surface representations

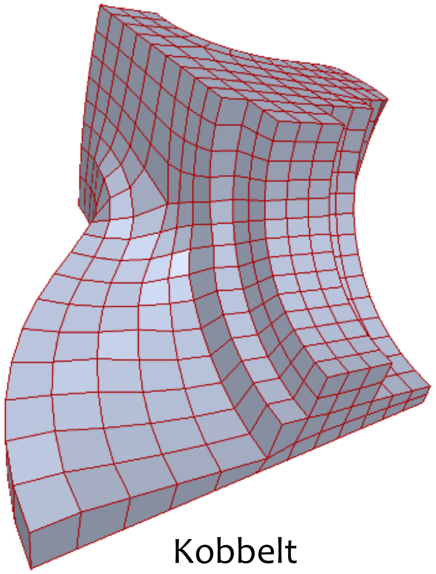
3D Model Representations

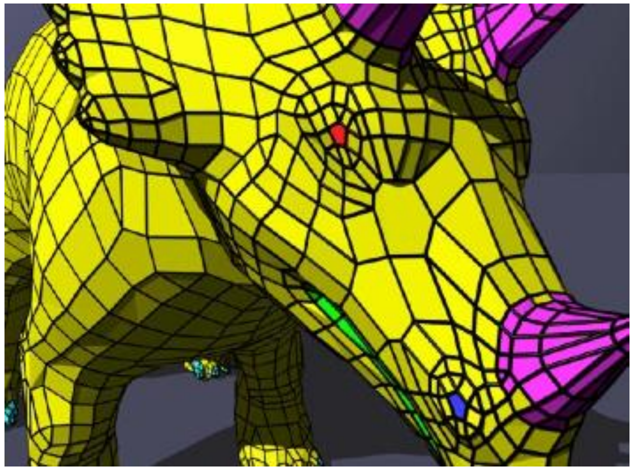
- Polygonal meshes
- Subdivision surfaces
- Parametric surfaces
- Implicit representations

Polygonal Meshes

- Discrete approximation of smooth surfaces
  - Piecewise-linear
  - Triangle meshes are the most popular


Garland

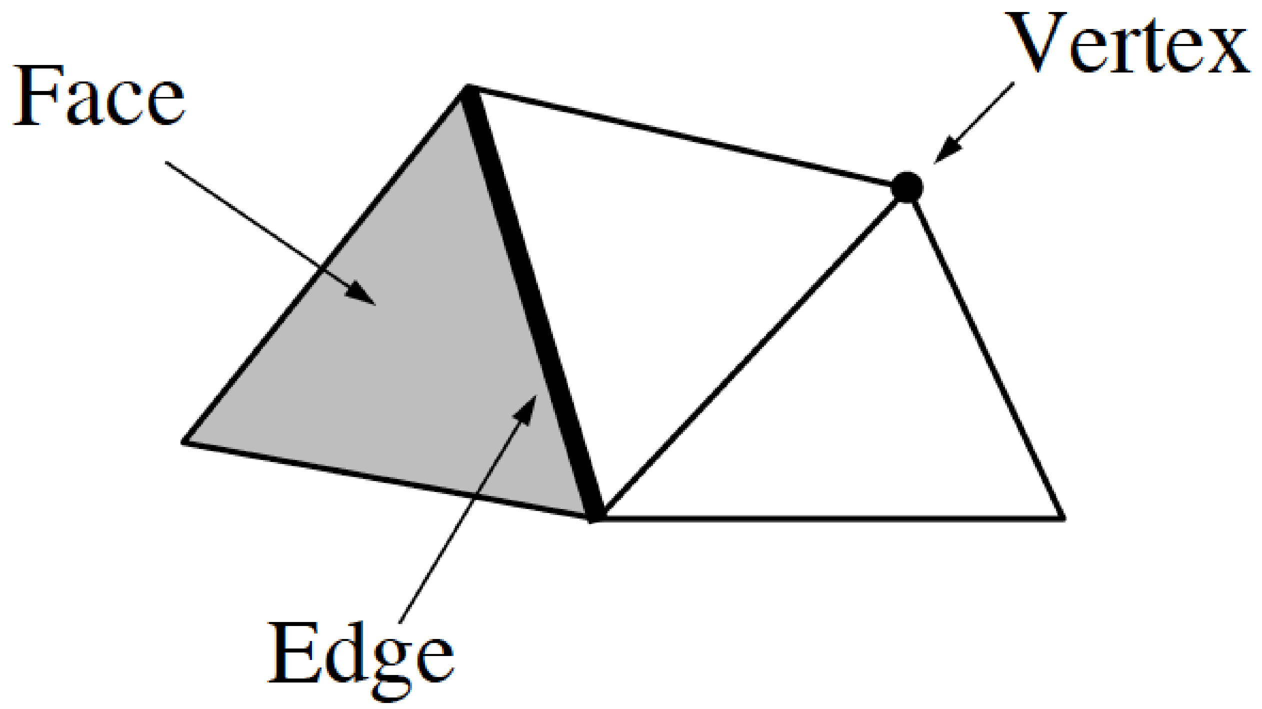

Kobbelt


Isenberg

Mesh Representations

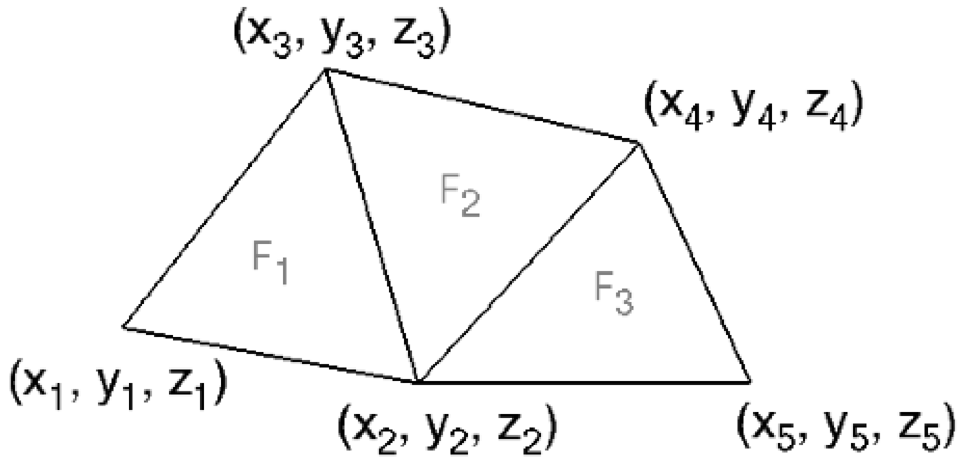
- Independent faces
- Vertex and face tables
- Adjacency lists
- Winged-Edge

Face, edge, vertex



Independent Faces

- Each face lists vertex coordinates
  - – Redundant vertices
  - – No topology information

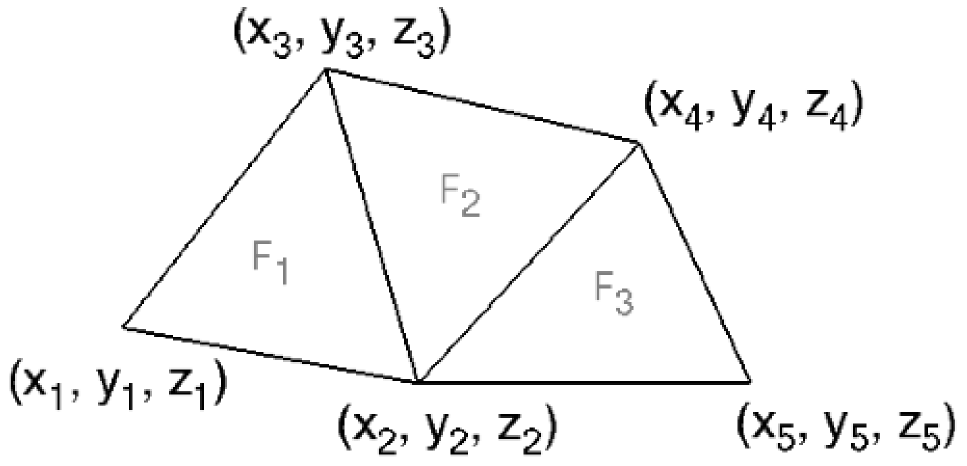


FACE TABLE

F_1	(x_1, y_1, z_1) (x_2, y_2, z_2) (x_3, y_3, z_3)
F_2	(x_2, y_2, z_2) (x_4, y_4, z_4) (x_3, y_3, z_3)
F_3	(x_2, y_2, z_2) (x_5, y_5, z_5) (x_4, y_4, z_4)

Vertex and Face Tables

- Each face lists vertex references
  - Shared vertices
  - Still no topology information

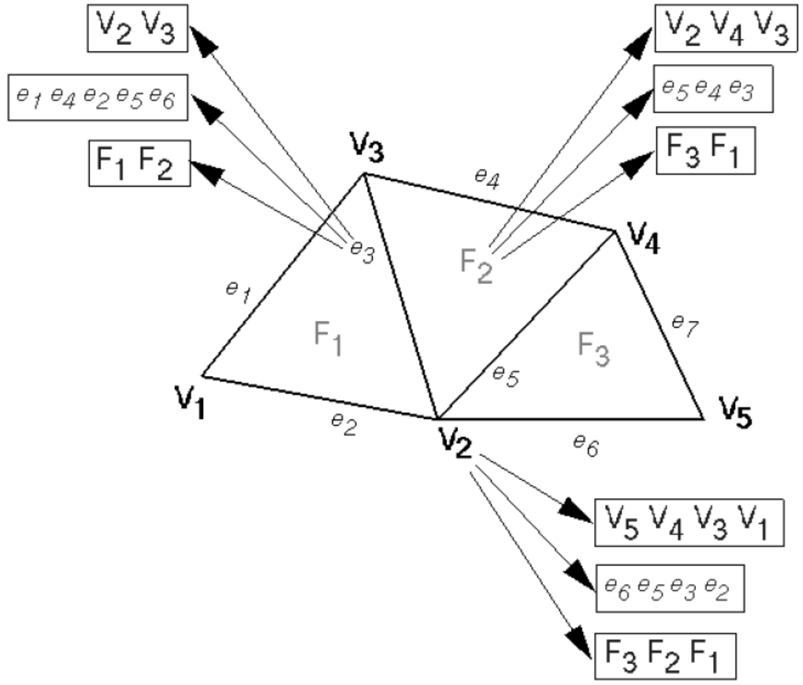


VERTEX TABLE			
V_1	X_1	Y_1	Z_1
V_2	X_2	Y_2	Z_2
V_3	X_3	Y_3	Z_3
V_4	X_4	Y_4	Z_4
V_5	X_5	Y_5	Z_5

FACE TABLE			
F_1	V_1	V_2	V_3
F_2	V_2	V_4	V_3
F_3	V_2	V_5	V_4

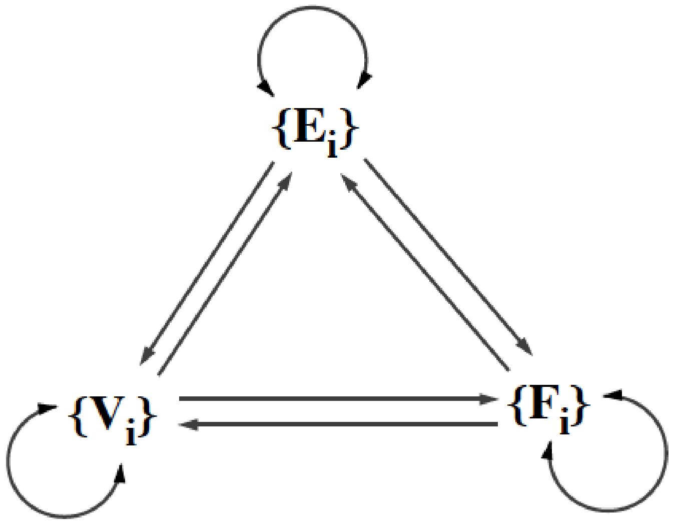
Adjacency Lists

- Store all vertex, edge, and face adjacencies
  - Efficient topology traversal
  - Extra storage



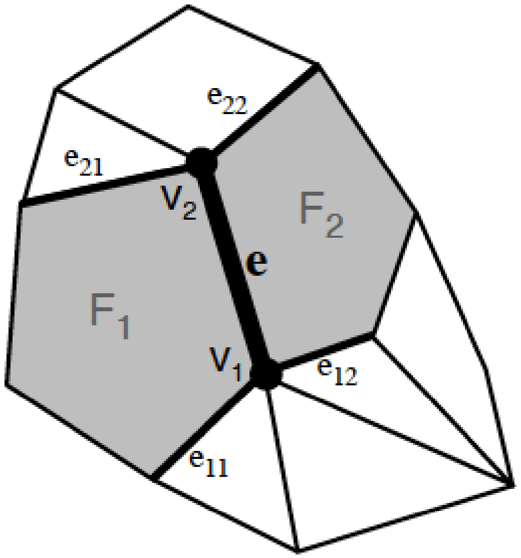
Can We Save Storage?

- Store some adjacency relationships and derive others
  - Partial adjacency list
- Which relations should be stored?



Winged Edge

- Adjacency stored in edges
  - Arrow: data to store
- Store each edge once

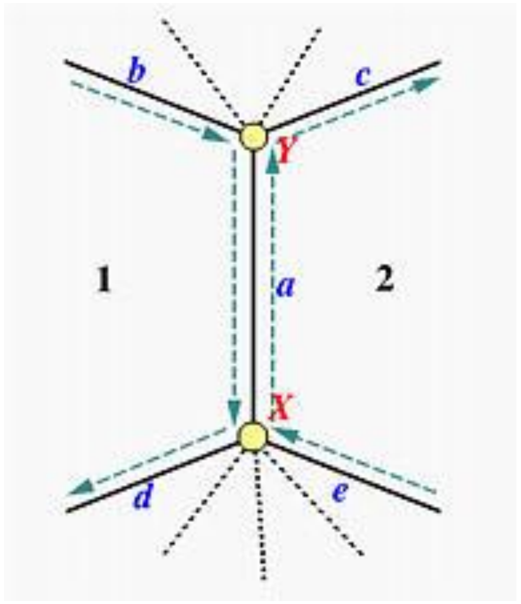


Funkhouser

$$\begin{tikzcd}
& \{E_i\} \arrow[loop above, "4"] \arrow[dl, shift right, "2"'] \arrow[dr, shift left, "2"] & \\
\{V_i\} \arrow[ur, shift right, "1"'] & & \{F_i\} \arrow[ul, shift left, "1"]
\end{tikzcd}$$

Winged Edge

- Clockwise direction

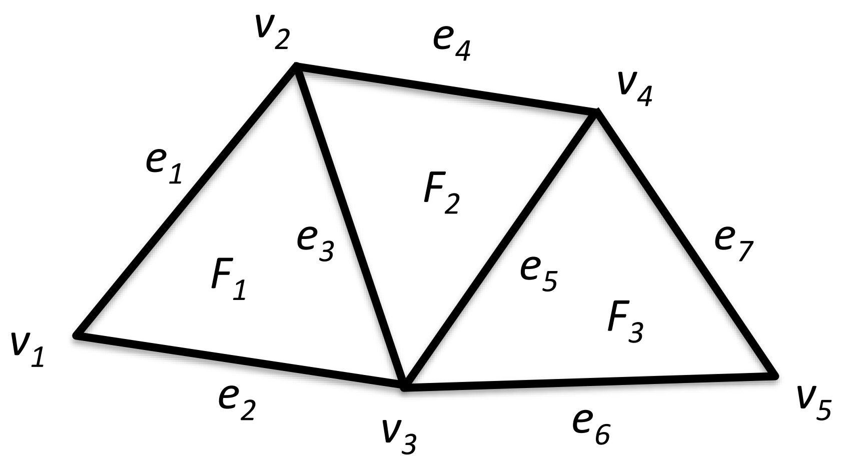


Shene

Edge Table

Edge	Vertices		Faces		Left Traverse		Right Traverse	
Name	**Start**	**End**	**Left**	**Right**	**Prev**	**After**	**Prev**	**After**
a	X	Y	1	2	b	d	e	c

Winged Edge Example



Edge	Vertices		Faces		Left Traverse		Right Traverse	
Name	**Start**	**End**	**Left**	**Right**	**Prev**	**After**	**Prev**	**After**
e1	v1	v2	.	F1	.	.	e2	e3
e3								

Winged Edge Implementation

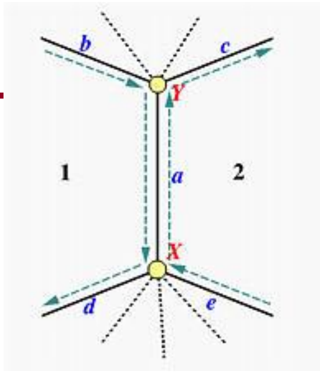
```
struct WE_edge
{
    WE_vert *v_start;
    WE_vert *v_end;
    WE_face *f_left;
    WE_face *f_right;
    WE_edge *e_left_prev;
    WE_edge *e_left_after;
    WE_edge *e_right_prev;
    WE_edge *e_right_after;
};
```

```
struct WE_face
{
    WE_edge *edge;
};

struct WE_vert
{
    float x;
    float y;
    float z;
    WE_edge *edge;
};
```

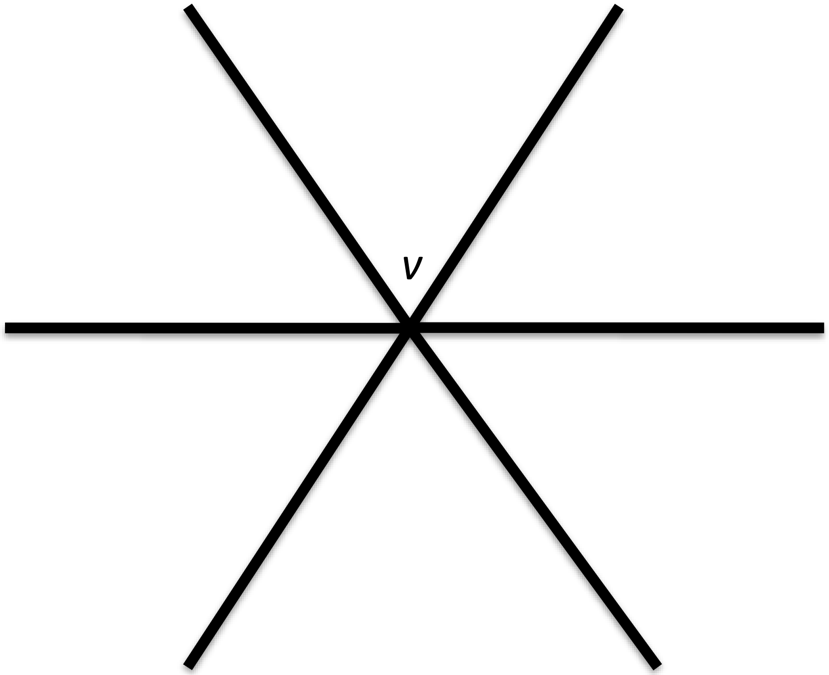
We assume that vertex stores the edge that starts from it.
We assume that face stores the edge whose right face is itself.

Traverse Neighbors

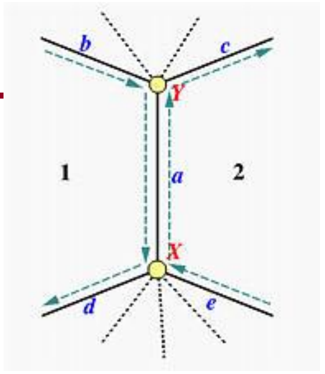
- 1-ring edges of a vertex v



```
edge = v->edge;  // first edge
start_edge = edge

--------

do {

  // counter-clock wise next edge
  if(edge->v_start == v) edge = edge->e_left_after
  else edge = edge->e_right_after

  // clockwise next edge
  if(edge->v_start == v) edge = edge->e_right_prev
  else edge = edge->e_left_prev

} while (start_edge != edge)
```

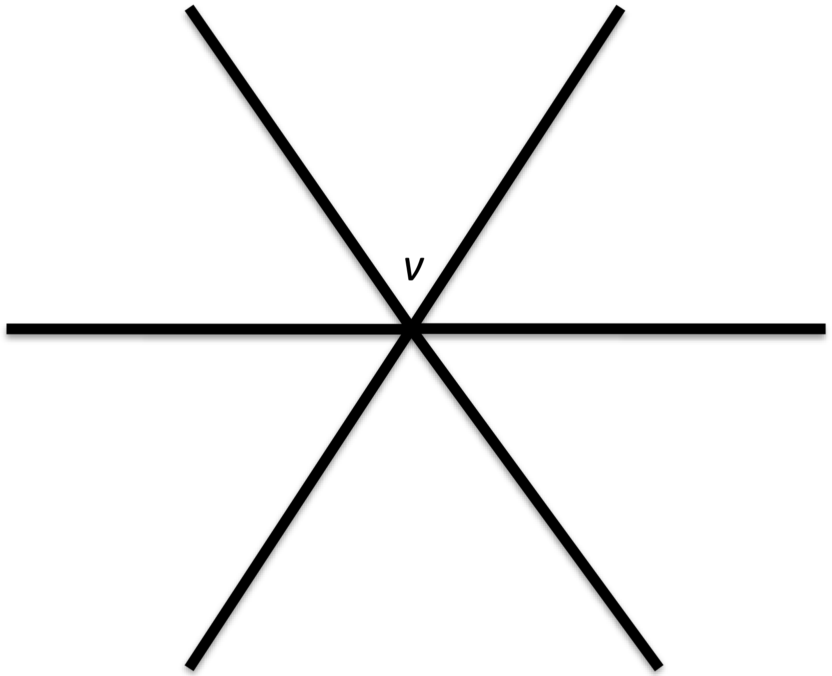


Traverse Neighbors



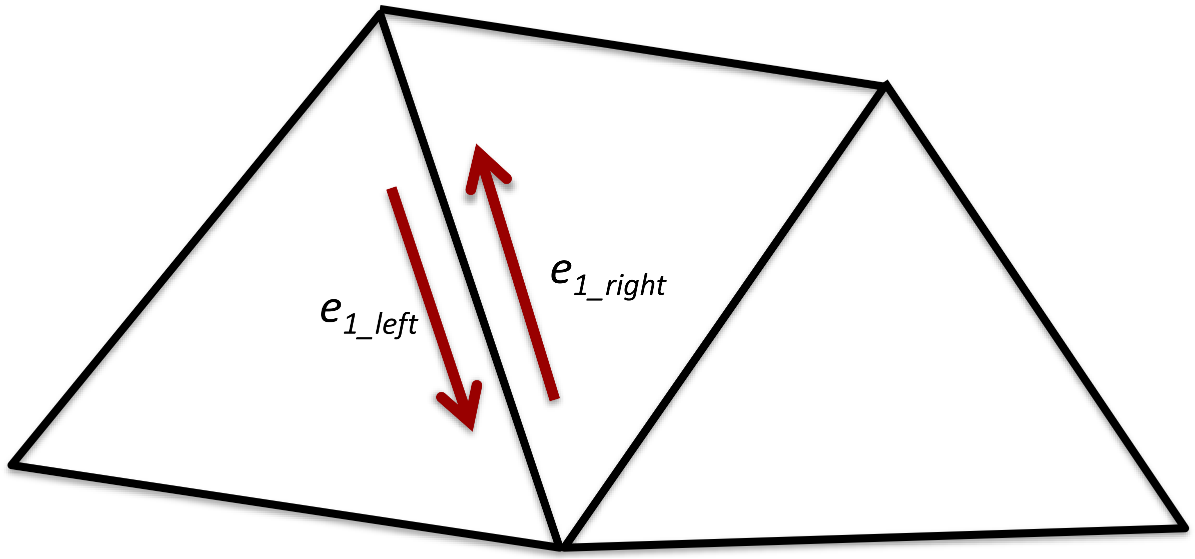
- 1-ring vertices of a vertex v
  - Clockwise traversal

```
edge = v->edge;  // first edge

vertex = edge->v_end // first vertex
start_vertex = vertex

--------

do {

   // clockwise next edge
   if(edge->v_start == v) edge = edge->e_right_prev
   else edge = edge->e_left_prev

   // vertex
   if(v != edge->v_start) vertex = edge->v_start
   else vertex = edge->v_end

} while (start_vertex != vertex)
```



Half Edge

- Adjacency stored in half edges
- Each edge is stored twice



Half Edge Implementation

```
struct HE_edge
{
    HE_vert *vert;
    HE_face *face;
    HE_edge *pair;
    HE_edge *next;
};
```

```
struct HE_face
{
    HE_edge *edge;
};


struct HE_vert
{
    float x;
    float y;
    float z;
    HE_edge *edge;
};
```

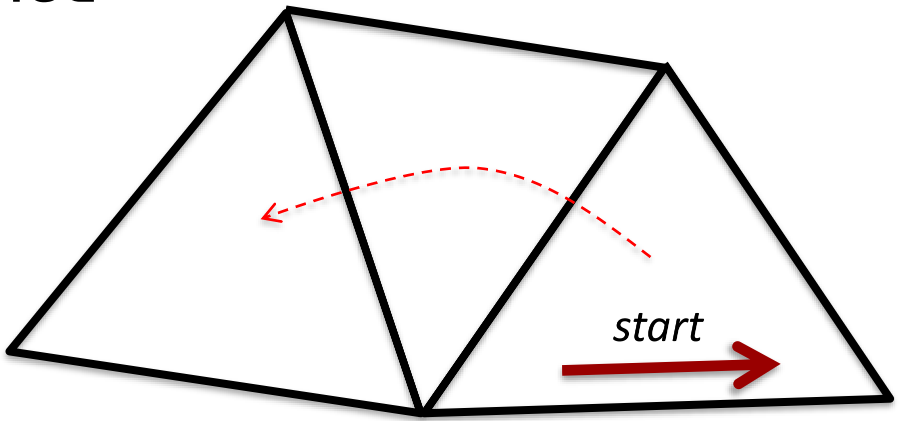
Same storage as WE but more efficient neighbor access
i.e., for a given edge, direction is always determined, but WE is not.

Half Edge Implementation

- When loading a mesh, we need to find a paired edge quickly
  - – hash map
- Algorithm
  - – For a given edge (u,v), check if edge (v,u) is already in the map
    - If yes, then pair (u,v) and (v,u) and delete (v,u) from the map
    - If not, add (u,v) in the map

Note about Half Edge

- If an edge is boundary, there is no pair half edge
- If a vertex is boundary, set one of the boundary edge as starting edge
  - When iterate neighbor vertices, stop if boundary vertex is reached



Questions?

Image courtesy of Marc Levoy