3D scanning

Lecture 12 – A

From (3D) synthetic to real

- In the CG part of the course, we have seen how to construct synthetic 3D worlds and render images of them
- We have constructed some simple 3D geometry (triangle, plane, cube, sphere) using geometric derivation for vertices position
- More complex geometries can be difficult to obtain synthetically and typically require advanced modeling software (see for example *Blender* https://www.blender.org/)

From (3D) synthetic to real

- Alternatively, 3D models can be acquired scanning 3D real objects/subjects
- These 3D models can then be imported and used for CG or directly processed for several other applications

From (3D) synthetic to real

- In this second part of the course, we will see how to
  - Acquire 3D objects using scanners, register them and obtain triangulated meshes
  - Represent these scans using appropriate geometric models
  - Perform computation on the object surface
  - Applications to 3D shape/object recognition, biometrics, behavior recognition, etc.
- We will use Matlab to load and process 3D models

3D ACQISITION

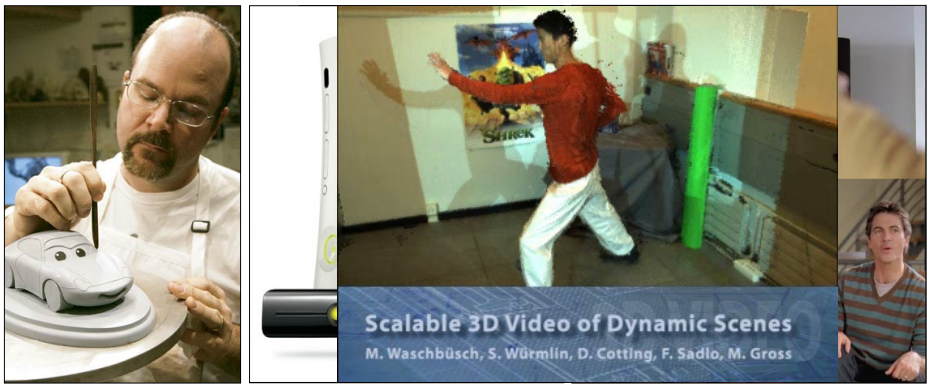
3D scanning

- A **3D scanner** is a device that analyses a real-world object or environment to collect data on its **shape** and possibly its **appearance** (e.g., color)
- The collected data can then be used to construct **digital 3D models**
- Collected 3D data is useful for a wide variety of applications
  - These devices are used extensively by the *entertainment industry* in the production of *movies* and *video games*
  - Other common applications of this technology include *industrial design*, *orthotics* and *prosthetics*, *reverse engineering* and *prototyping*, *quality control/inspection* and *documentation of cultural artifacts, biometrics applications*

Applications

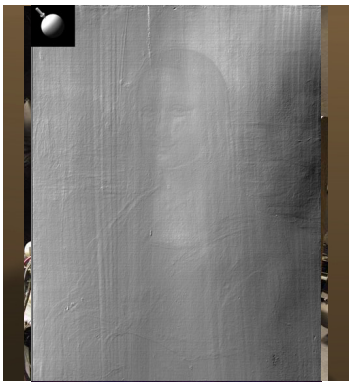
- 3D scanners are used by the **entertainment industry** to create **digital 3D models** for movies, video games and leisure purposes
- They are heavily utilized in **virtual cinematography**. In cases where a real-world equivalent of a model exists, it is much faster to scan the real-world object than to manually create a model using 3D modeling software
- Frequently, artists sculpt physical models of what they want and scan them into digital form rather than directly creating digital models on a computer

Applications: Entertainment and consumer applications



- Import sculptures into a 3D modeling/rendering pipeline
- Capture geometric (and photometric) properties for relighting
- Fit clothes, track 3D interaction, free-viewpoint video (3D TV), etc.

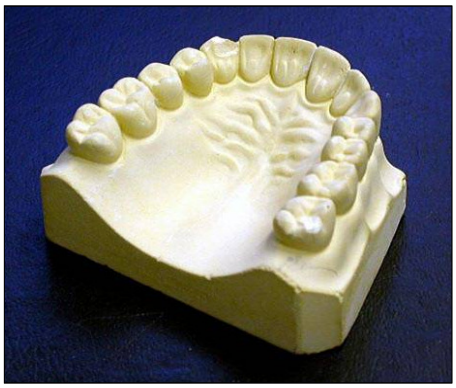
Applications: Historical preservation

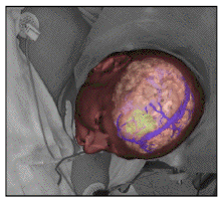


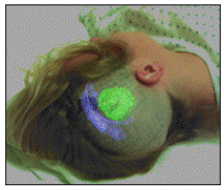
- Preserve/restore deteriorating works and unite dispersed collections
- Facilitate academic study (tooling, lighting, revision history)
- Replicate collections (souvenirs, retain repatriated works, etc.)

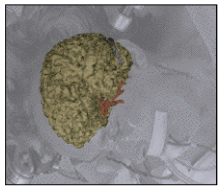
9

Applications: Medical imaging and surgical planning





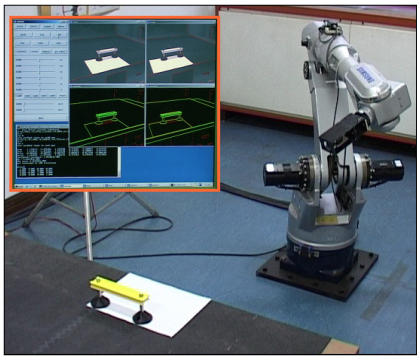




- Medical imaging (X-ray, CT, MRI, etc.) and surgical planning
- Measuring dimensions (dental impressions and hip replacement surgery)
- Tele-surgery (augmented virtual reality, video see-through, etc.)

10

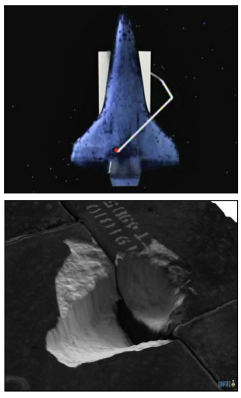
Applications: Robotics (interaction and navigation)

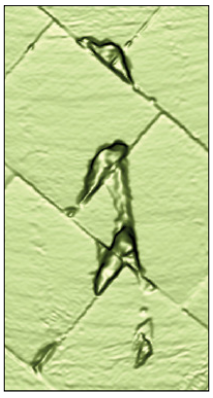


- Motion planning (manipulation, gripping, pushing/pulling, etc.)
- Simultaneous localization and mapping (SLAM)
- Autonomous navigation (DARPA Grand/Urban Challenge)

11

Applications: Inspection and reverse engineering





- Manufacturing and process control (tolerances and alignment)
- Reverse engineering (repairing antiques and replicating designs)
- Remote inspection (inaccessible or dangerous environments)

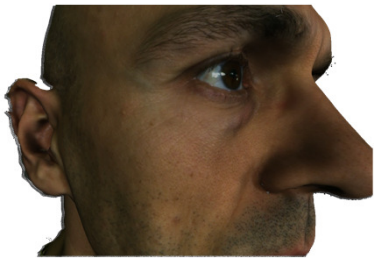
12

3D scanning

- Many different technologies can be used to build 3D-scanning devices; each technology comes with its own limitations, advantages and costs
- Many limitations in the kind of objects that can be digitized are still present
  - For example, **optical technologies** encounter many difficulties with **shiny**, **mirroring** or **transparent** objects
- For example, industrial computed tomography scanning can be used to construct digital 3D models, applying non-destructive testing

3D scanning

- The purpose of a 3D scanner is usually to create a **point cloud** of geometric samples on the surface of the object/subject
- These points can then be used to extrapolate (**reconstruct**) the shape
- If color information is collected at each point, then the colors on the surface can also be determined



3D scanning

- 3D scanners share several common traits with cameras
- Like most cameras, they have a cone-like field of view, and like cameras, they can only collect information about **surfaces that are not obscured**
- While a camera collects **color information** about surfaces within its field of view, a 3D scanner collects **distance information** about surfaces within its field of view
- The "picture" produced by a 3D scanner describes the **distance** to a **surface at each point in the picture**
- This allows the three dimensional position of each point in the picture to be identified

3D scanning

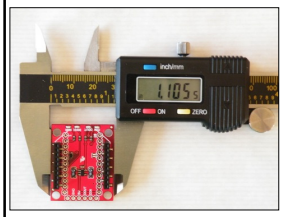
- For most situations, a single scan will not produce a complete model of the object/subject
- Multiple scans, even hundreds, from many different directions are usually required to obtain information about all sides of the object/subject
- These scans have to be brought into a common reference system, a process that is usually called **alignment** or **registration**, and then merged to create a complete model
- In lecture 13, we will see more about 3D rigid and non-rigid registration using the ***Iterative Closest Point*** (ICP) and ***Coherent Point Drift*** (CPD) algorithms

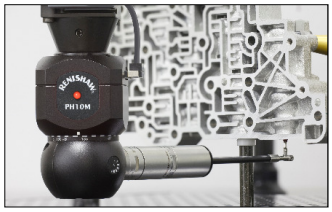
Technologies

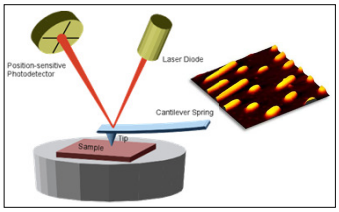
- There are a variety of technologies for digitally acquiring the shape of a 3D object
- A well established classification divides them into two types
  - **Contact**
  - **Non-contact**
    - **Passive**
    - **Active**
- There are a variety of technologies that fall under each of these categories

Contact

- Contact 3D scanners probe the object through **physical touch**, while the object is in contact with or resting on a precision flat surface plate, ground and polished to a specific maximum of surface roughness
- Where the object to be scanned is not flat or can not rest stably on a flat surface, it is supported and held firmly in place by a fixture







Contact

- The scanner mechanism may have three different forms
  - A **carriage system** with rigid arms held tightly in perpendicular relationship and each axis gliding along a track. Such systems work best with flat profile shapes or simple convex curved surfaces
  - An **articulated arm** with rigid bones and high precision angular sensors. The location of the end of the arm involves complex math calculating the wrist rotation angle and hinge angle of each joint. This is ideal for probing into crevasses and interior spaces with a small mouth opening
  - A **combination** of both methods may be used, such as an articulated arm suspended from a traveling carriage, for mapping large objects with interior cavities or overlapping surfaces

Contact

- A **Coordinate Measuring Machine** (CMM) is an example of a contact 3D scanner
- It is used mostly in manufacturing and can be very precise

Contact

- The disadvantage of CMMs though, is that it requires contact with the object being scanned
- Thus, the act of scanning the object might modify or damage it
  - This fact is very significant when scanning delicate or valuable objects such as historical artifacts
- The other disadvantage of CMMs is that they are relatively **slow** compared to the other scanning methods
  - Physically moving the arm that the probe is mounted on can be very slow

Non-contact active scanners

- Active scanners emit some kind of **radiation** or **light** and detect its **reflection** or **radiation passing through** object in order to probe an object or environment
- Possible types of emissions used include light, ultrasound or x-ray

- Main technologies are
  - **Time-of-flight** (ToF)
  - **Triangulation** (stereo vision)

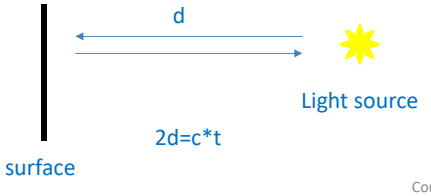
NON-CONTACT ACTIVE SCANNERS: TIME OF FLIGHT (TOF)

Time of flight

- The **time-of-flight** (ToF) 3D laser scanner is an active scanner that uses laser light to probe the subject
- At the heart of this type of scanner is a **ToF laser range finder**
- The laser range finder finds the distance of a surface by timing the **round-trip time of a pulse of light**
- A laser is used to emit a pulse of light and the amount of time before the reflected light is seen by a detector is measured

Time of flight

- Since the speed of light *c* is known, the round-trip time determines the travel distance of the light, which is twice the distance between the scanner and the surface
- If *t* is the round-trip time, then **distance** is equal to `c*t/2`
- The accuracy of a time-of-flight 3D laser scanner depends on **how precisely** we can **measure** the **time** *t*
  - 3.3ps is (approximately) the time taken for light to travel 1mm



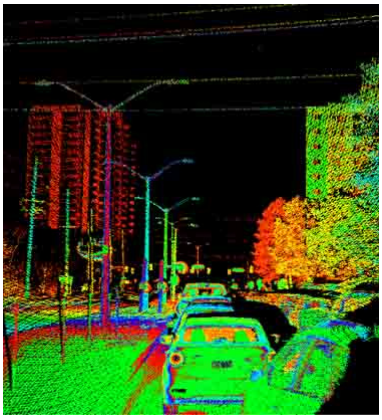
Time of flight

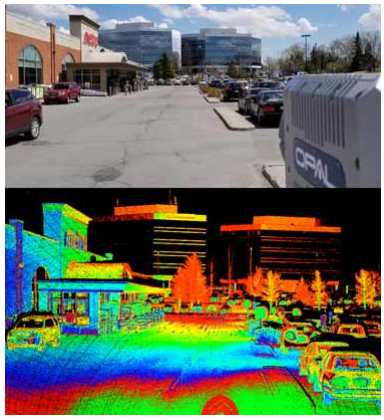
- The **laser range finder** only detects the distance of **one point** in its direction of view
- Thus, the scanner scans its entire field of view **one point at a time** by changing the range finder's direction of view to scan different points
- The view direction of the laser range finder can be changed either by rotating the range finder itself, or by using a **system of rotating mirrors**
- The latter method is commonly used because mirrors are much lighter and can thus be rotated much faster and with greater accuracy
  - Typical time-of-flight 3D laser scanners can measure the distance of 10,000~100,000 points every second

Time of flight

- Example of ToF is **LiDAR** (**Light Detection And Ranging**) that measures distance by illuminating a target with a laser light
- Lidar is popularly used as a technology to make high-resolution maps, with applications in *geodesy*, *geomatics*, *archaeology*, *geography*, *geology*, *geomorphology*, *seismology*, *forestry*, *atmospheric physics, airborne laser swath mapping* (ALSM) and *laser altimetry*
- What is known as LiDAR is sometimes simply referred to as laser scanning or 3D scanning, with terrestrial, airborne and mobile applications

Time of flight





Lidar: application to automotive

ToF cameras

- Time-of-flight devices are also available in a 2D configuration. This is referred to as a **time-of-flight camera**
- A **ToF camera** is a range imaging camera system that resolves distance based on the known speed of light, measuring the **ToF** of a **light signal** between the **camera** and the **object/subject for each point of the image**
- The ToF camera is a class of *scannerless* LiDAR, in which the **entire scene is captured with each laser or light pulse**, as opposed to point-by-point with a laser beam such as in scanning LiDAR systems

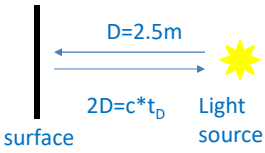
ToF cameras

- ToF camera products for civil applications began to emerge around 2000, as the semiconductor processes became fast enough for such devices
- The systems cover ranges of a few centimeters up to several kilometers
- The **distance resolution** is about 1 cm. The lateral resolution of ToF cameras is generally low compared to standard 2D video cameras
- Compared to 3D laser scanning methods for capturing 3D images, **ToF cameras operate very quickly**, providing up to 160 images per second

ToF cameras

- The simplest version of a ToF camera uses **light pulses** or a **single light pulse**
- The **illumination is switched on for a very short time**, the resulting **light pulse illuminates the scene** and is **reflected by the objects in the field of view**
- The **camera lens gathers the reflected light** and **images it onto the sensor** or focal plane array
- Depending upon the distance, the **incoming light experiences a delay**
- As light has a speed of approximately $c = 3*10^8$ m/s, this delay is **very short**

ToF cameras



Example

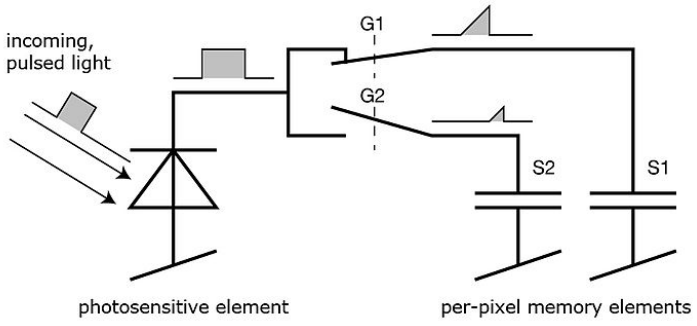
- An object 2.5m away will delay the light by
  $t_D = 2*D/c = 2*2.5/3*10^8 = 1.6*10^{-8}s = 16.66ns$
- For amplitude modulated arrays, the **pulse width** of the **illumination** determines the **maximum range the camera can handle**
- With a pulse width of, e.g., 50ns, the range is limited to
  $D_{max} = ½*c*t_0 = ½*3*10^8 * 50*10^{-9} = 7.5m$
- These short times show that the **illumination unit** is a **critical part** of the system
  - Only with **special LEDs** or **lasers** it is possible to generate such short pulses

ToF cameras

- The single **pixel** consists of a **photo sensitive element** (e.g., a **photo diode**) that **converts** the **incoming light** into a **current**
  - In **analog timing imagers**, connected to the photo diode are fast switches, which direct the **current to one of two** (or several) **memory elements** (e.g., a **capacitor**) that act as **summation elements**
  - In **digital timing imagers**, a **time counter**, that can be running at several gigahertz is connected to each photo-detector pixel and **stops counting when light is sensed**

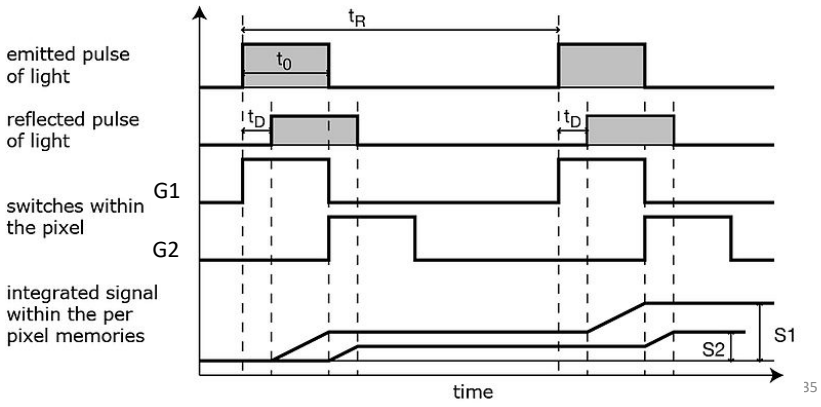
ToF cameras

- In the diagram of an **amplitude modulated array analog timer**, the pixel uses **two switches** (G1 and G2) and two **memory elements** (S1 and S2)



ToF cameras

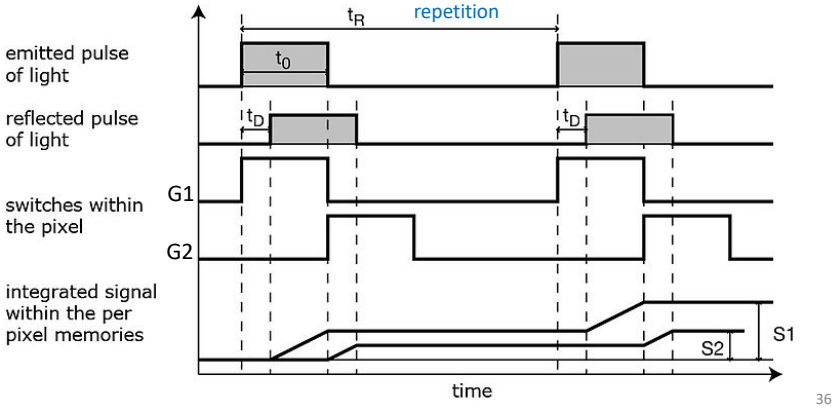
- The **switches are controlled** by a **pulse** with the **same length as the light pulse**, where the control signal of switch G2 is delayed by exactly the pulse width



35

ToF cameras

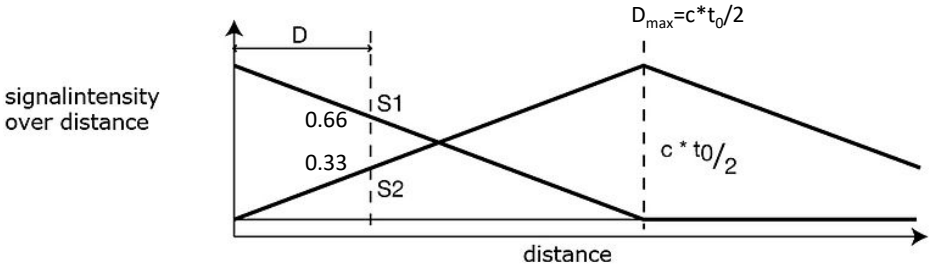
- **Depending on the delay**, only part of the light pulse is sampled through G1 in S1, the other part is stored in S2



36

ToF cameras

- **Depending on the distance**, the ratio between S1 and S2 changes as depicted in the drawing



  - Considering the previous example with pulse width of 50ns, because only small amounts of light hit the sensor within 50ns, not only one, but **several thousand pulses are sent out** (repetition rate t_R) and gathered, thus **increasing the signal to noise ratio**

ToF cameras

- After the exposure, the pixel is read out and the following stages measure the signals S1 and S2
- As the **length of the light pulse is defined**, the distance can be calculated with the formula

  $D = ½ * c * t_0 * S2/(S1+S2)$

- In the example, the signals have values S1 = 0.66 and S2 = 0.33. The distance is therefore

  D = 7.5m * 0.33/(0.33+0.66) = 2.5m

ToF cameras

- In the presence of **background light**, the memory elements receive an additional part of the signal. This would disturb the distance measurement
  - To eliminate the background part of the signal, the whole **measurement can be performed a second time** with the **illumination switched off**
- If the objects are further away than the distance range, the result is also wrong
  - Here, a second measurement with the control signals delayed by an additional pulse width helps to suppress such objects
- Other systems work with a **sinusoidally modulated** light source instead of the pulse source

ToF cameras

- Based on this idea ***Canesta Vision*** developed ***Canesta*** [3]
- It works by modulating the outgoing beam with an RF carrier, then measuring the **phase shift** of that carrier on the receiver side
  - This approach has a modular error challenge; ranges are mod the maximum range, which is the RF carrier wavelength
- ***CanestaVision*** was purchased by ***Microsoft*** in 2010
  - The ***Kinect for Xbox One*** (***Kinect 2***) was based on ToF technology from *Canesta*
- This technology is illustrated in the following with some detail

Kinect 2

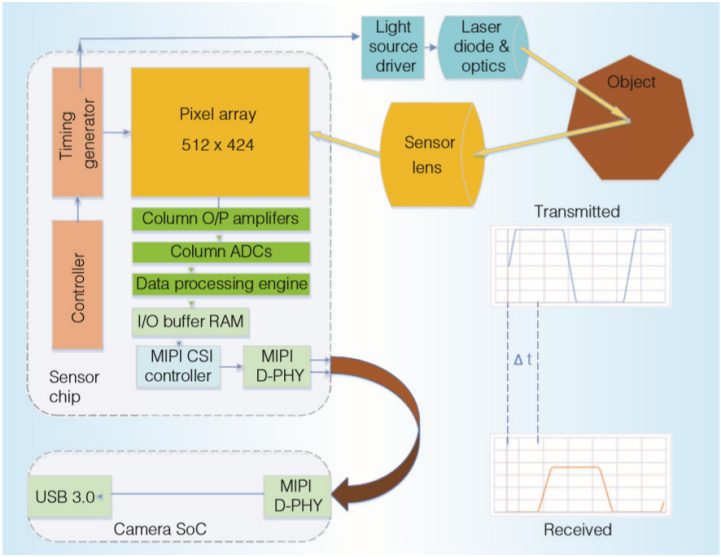
- Characteristics
  - depth image (512 x 424 pixel)
  - RGB image (1.920 x 1.080 pixel – 30 fps)
  - ToF sensor
  - Range from 0.4 to 8 meters

Frame #90 Frame #130 Frame #170

ToF depth sensor

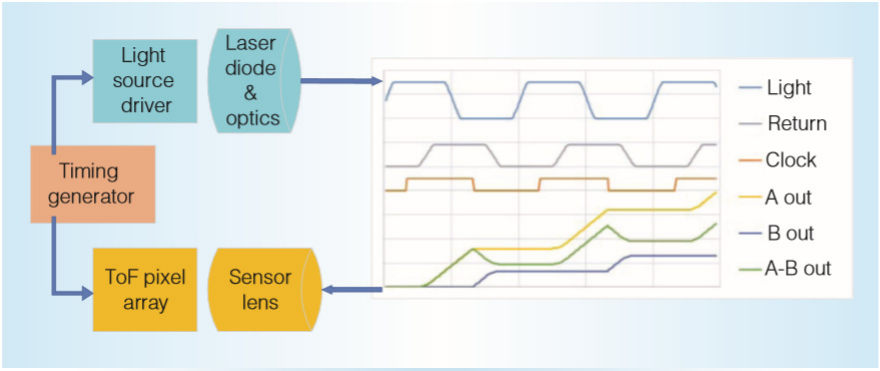
- A typical ToF sensor consists of a modulated light source such as a **laser** or **LED**, an **array of pixels**, each capable of detecting the **phase of the incoming light**, and an ordinary **optical system** for focusing the light onto the sensor
- The light is given a **modulation envelope** by **rapidly turning the light source on and off**
- Distance measurement is achieved by **measuring the phase** of the **modulation envelope** of the **transmitted light as received** at the **pixel array**

ToF depth sensor



43

ToF depth sensor



CMOS sensor chip

- The key part of the sensor design is the special pixel structure (a cross-section is shown)
- The differential structure accumulates photo-generated charges in two collection nodes using two modulated gates

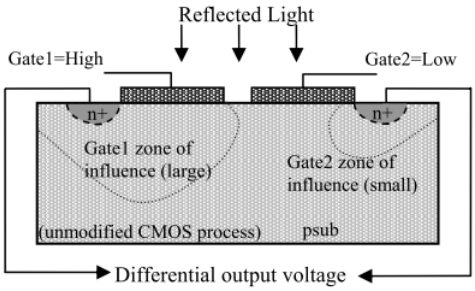


Image from Canesta's paper

- The gate modulation signals are synchronized with the light source, and hence depending on the phase of incoming light, one node collects more charges than the other

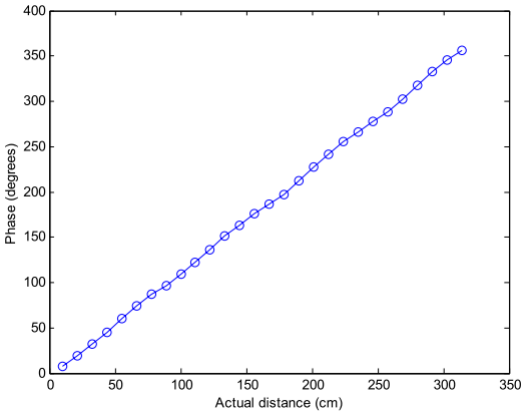
CMOS sensor chip

- At the end of **integration**, the **voltage difference between the two nodes** is read out as a measure of the **phase of the reflected light**
- Thus, in effect, this pixel simultaneously performs the function of **mixing** and **low pass filtering**

CMOS sensor chip

- A **linear relationship** exists between the **phase values** and **actual distance**

Image from Canesta's paper



Phase-distance relationship for the depth sensor

Analysis of resolution

- An important attribute of a depth sensor is its **depth resolution**
- Here we analyze the factors that affect the resolution of the system
- For simplicity of the analysis, we assume that the pixel output is **single-ended voltage** even though it is actually differential

Analysis of resolution

- Let P_{laser} be the **optical power of the light source**, and A the total (target) **area** illuminated
- The amount of laser light received by the sensor depends on the **reflectivity of the objects in the direction of the sensor**
  - This **reflectivity** is denoted by r
- The total number of electrons generated in one pixel at the end of an **integration** (shutter) **period** of T can be written as

$$N_{electrons} = P_{laser}\ k_{opt}\ q_e\ r\ T\ /\ A$$

  - where q_e is the **quantum efficiency** (i.e., percentage of photons hitting the device's photoreactive surface that produce charge carriers) and k_{opt} is a constant determined by the properties of the optical system including lenses, diffuser, pixel size, etc.

Analysis of resolution

- The electrons collected by the pixel are stored in a **storage capacitor** C, whose **voltage** at the end of the **integration period** is the **signal produced by the pixel** as a **measure of phase delay** (distance)
- Due to the **interaction** of the **reflected light** with the **reference** (modulation) signal in the pixel, only a **fraction** p of the electrons contribute to this signal
- Essentially p represents the **phase overlap** between the **reflected light** and the **reference signal**

Analysis of resolution

- Hence the **voltage** across the **storage capacitor** at the end of integration is

  $V_{signal} = qpN_{electrons}/C = qpP_{laser}k_{opt}q_e rT/CA$

  where q is the **charge** of an electron, 1.6×10^{-19} *Coulomb*

- The value of p at a particular pixel depends on the **distance** between this pixel and the target it is imaging
- Depending on the distance, and hence p, V_{signal} changes from $V_{signal}(p=0)$ to $V_{signal}(p=1)$
- Since $V_{signal}(p=0)=0$, the total voltage swing V_{swing} is equal to $V_{signal}(p=1)$

Analysis of resolution

- One of the components of noise on V_{signal} is the **shot noise**
- There are other sources of random or pseudo random noise as well. These include **ADC quantization noise**, **kTC reset noise**, **thermal noise**, etc.
- The **shot noise** is usually the dominant source of error and determines the **accuracy of depth measurements**
- The error in V_{signal} due to shot noise is calculated by projecting the uncertainty in the number of electrons to the voltage across the storage capacitor
- In turn, this results in an RMS (root mean square) **voltage error** of

$$V_{noise} = \frac{q\sqrt{pN_{electrons}}}{C} = \frac{q}{C}\sqrt{\frac{pP_{laser}k_{opt}q_e rT}{A}}$$

Analysis of resolution

- The ***voltage resolution*** of the sensor is calculated by V_{swing}/V_{noise} , which depends on p in V_{noise}
- For resolution analysis, we use $V_{noise}(p=1)$ since it maximizes the magnitude of noise
- Thus, from the voltage resolution, the ***depth resolution*** is determined by the number of small divisions that the **unambiguous range** (i.e., maximum distance without considering ambiguous repetitions) can be reliably divided

$$resolution = \frac{Range}{V_{swing}/V_{noise}(p=1)} = \frac{c}{2f_m}\sqrt{\frac{A}{P_{laser}k_{opt}q_e rT}}$$

Analysis of resolution

- The finite resolution implies that the **distance value measured by each pixel deviates from the correct value** by an **amount whose statistical standard deviation is given by the resolution equation**
- Changing each parameter in the resolution equation involves a tradeoff
- For instance, a **high power laser** results in **better resolution**, but **increases the electrical power consumption and the cost of the system**
- Resolution can also be improved by **reducing the imaged area**, or by **increasing the target reflectivity** *r*, or by **increasing the integration time**, *T*

Analysis of resolution

- The integration time cannot be increased arbitrarily, however; since it **determines the frame rate** for which most applications have a **minimum requirement**
- It is also possible to **improve resolution** with q_e by using a **lower-wavelength laser**
- However, keeping the **wavelength in the infrared range** is desired by most applications
- Another way of improving resolution is to **increase the modulation frequency** f_m; although this **reduces the unambiguous range**, resulting in a **higher degree of aliasing**

Analysis of resolution

- In the presence of **ambient light** the above resolution equation needs to be revised since **ambient light contributes to the** (**shot**) **noise**, but not to the useful signal
- Let P_{amb} be the ambient light power present in the target area A
- Factoring in P_{amb}, we find that the RMS error in the voltage across the storage capacitor becomes

$$V_{noise-amb} = \frac{q}{C}\sqrt{\frac{p(P_{laser} + P_{amb})k_{opt}q_e rT}{A}}$$

Analysis of resolution

- Since the useful signal V_{signal} remains the same, the depth resolution changes to

$$resolution = \frac{c}{2 f_m} \sqrt{\frac{P_{laser} + P_{amb}}{P_{laser}^2} \frac{A}{k_{opt} q_e r T}}$$

- Clearly, to get the best resolution we **must minimize** P_{amb}
- Thus, for the sensor to operate in the presence of sunlight some techniques have to be developed

Issues

- There are some known issues that affect ToF acquisitions

Noise behavior

- The resolution analysis given above is a **per-pixel, per-frame resolution**. The effect of limited resolution is seen as a noisy behavior on the depth images, where the standard deviation of the noise is equivalent to resolution of the system. The noise can be reduced using techniques from radar sensing, in particular through spatial and temporal averaging

Ambient effect

- Ambient light is considered as unwanted light that has the same wavelength as the light source of the system. Although ambient light is non-modulated, constant light, it has a **shot noise component** that causes errors in the sensor. As a consequence, there may be noisy data in the area of the image receiving ambient light

Issues

Motion artifacts

- For **higher accuracy**, it is necessary to have the light signal switch between **multiple phases** within a **single frame**. However this approach may introduce **motion artifacts**. The effect is somewhat equivalent to having motion blur in a picture when a long exposure time is used. There is a similar motion artifact if there is fast movement in the scene. When the **object moves fast between the phases within a frame**, **each phase may potentially receive light coming from a different distance**
- The motion artifact is observed mostly around the edges of a moving object. There are various ways to solve this problem. An obvious method is to **increase the frame rate**. Alternatively one can estimate the motion in 3D, and then correct the range values around the boundaries. Yet another option is to do edge detection followed by elimination of motion artifacts around the edges

Aliasing problem

- Due to the **limited unambiguous distance range**, **aliasing** arises as an issue
- **Target objects** that are **separated by one full range** or **integer multiples of the full range**, are indistinguishable
  - For example, given a 50MHz **modulation frequency**, whose **unambiguous range is** 3m (D_{max} = c/2f = $3*10^8/2*50*10^6$= $3*10^8/10^8$=3m), an object at 1m and another at 4m from the sensor produce the **same range value**
  - Lowering the modulation frequency to, say, 10MHz increases the unambiguous range to 15m (D_{max} = c/2f = $3*10^8/2*10*10^6$= $3*10^8/0.2*10^8$=15m). However, the **resolution goes down** by a **factor of 5**, as seen from the previous resolution equation

$$resolution = \frac{c}{2 f_m} \sqrt{\frac{P_{laser} + P_{amb}}{P_{laser}^2} \frac{A}{k_{opt} q_e r T}}$$

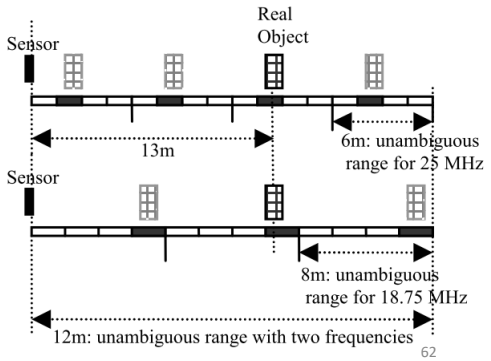
Aliasing problem

- One method of **increasing the unambiguous distance range** is to take **two or more measurements**, each with a **different modulation frequency**
- Suppose **two distance measurements** are made with **two frequencies** f_1 and f_2
- Let R_1 and R_2 be the **maximum unambiguous ranges** for these frequencies
- With two measurements, the effective unambiguous range is increased to the LCM (**least common multiple**) of R_1 and R_2

Aliasing problem

Example

- Object at 13m from the sensor. If we only use one modulation frequency of 25Mhz, we can infer that the object is either at 1m or 7m or 13m or 19m (D_{max} = c/2f = $3*10^8/2*25*10^6$= $3*10^8/5*10^7$=6m)
- If we only use 18.75Mhz for modulation, we can infer that the object is either at 5m or 13m or 21m (D_{max} = $3*10^8/37.5*10^6$=8m)
- Combining the two results, we conclude that the object must be at 13m

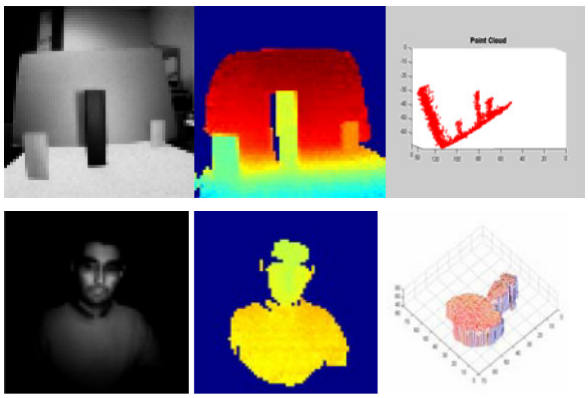


Aliasing problem

- Of course, this object would still be confused with another at 37m (13m+24m), but, compared to the one-frequency case, the **unambiguous range is extended significantly**
- One strategy in selecting modulation frequencies is to **maximize the LCM** of the corresponding ranges by **choosing two frequencies close to each other**. Another strategy is to use a log-based approach where a number of measurements are made with frequencies *f, 2*f, 4*f, 8*f* and so on
  - In each successive measurement the resolution is doubled while the range is halved
- By combining the results, an accurate value of distance over a long unambiguous range can be obtained

Acquisition examples

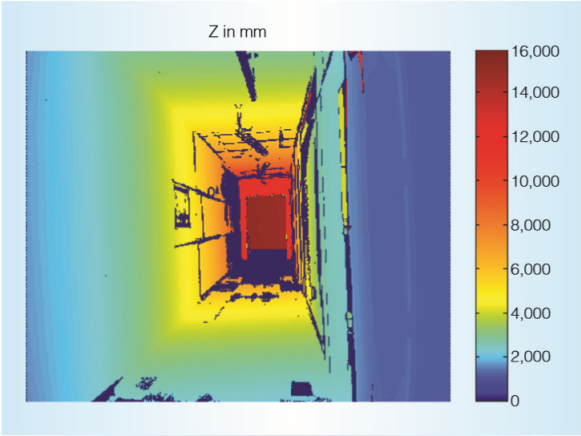
- The result of the acquisition process is a depth (range) image, where a gray or color level can be used to represent the depth
- Acquisition examples from [3]



Examples (Kinect)

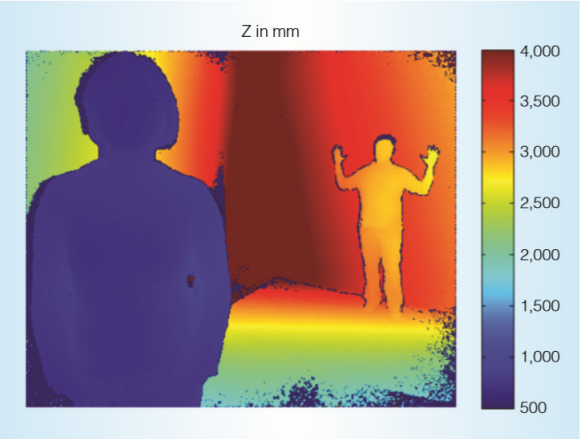
Depth image captured at a distance of 2.5 meters. The fine detail of facial features and folds in the clothing are clearly visible

Examples (Kinect)



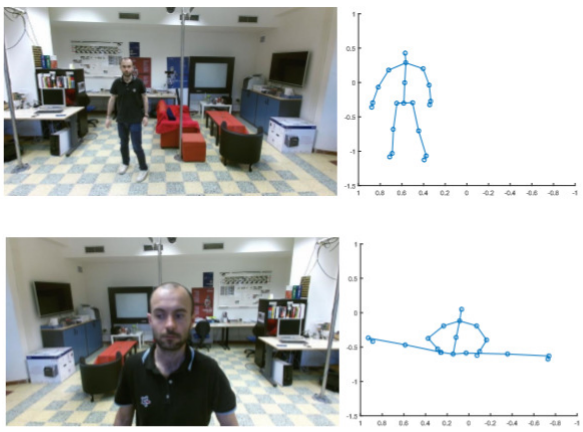
Depth range (denoted by a color map) looking down a long corridor. The depth changes smoothly along the wall, well beyond 10 m, without any trace of aliasing due to phase-wrap

Examples (Kinect)



Dynamic range figure recognition. The system captures figures close to the camera and far from the camera clearly

Face and skeleton (Kinect)



We will see how the skeleton is computed from the depth for Kinect in lecture 12-B

Note on Kinect

- In October 2017, Microsoft announced that the production of Kinect was discontinued
- In January 2018 the production of the Kinect Adapter was also discontinued
- However, in May 2018, Microsoft announced the new project **Kinect for Azure.** Kinect revived as cloud-powered sensor for HoloLens and more https://azure.microsoft.com/en-us/campaigns/kinect/

New Kinect with 1024x1024 depth resolution

ToF advantages

Simplicity

- In contrast to **stereo vision** or **triangulation systems**, the whole system is very compact: the illumination is placed just next to the lens, whereas the other systems need a certain minimum base line
- In contrast to **laser scanning systems**, no mechanical moving parts are needed

ToF advantages

Efficient distance algorithm

- It is a direct process to extract the distance information out of the output signals of the ToF sensor
- As a result, this task uses only a small amount of processing power, again in contrast to stereo vision, where **complex correlation algorithms** are implemented
- After the distance data has been extracted, object detection, for example, is also a straightforward process to carry out because the algorithms are not disturbed by patterns on the object

ToF advantages

Speed

- Time-of-flight cameras are able to measure the distances within a complete scene with a **single shot**
- As the cameras reach up to 160 frames per second, they are ideally suited to be used in **real-time applications**

ToF disadvantages

Background light

- When using CMOS or other integrating detectors or sensors that use **visible** or **near infra-red light** (400 nm - 700 nm), although most of the background light coming from artificial lighting or the sun is suppressed, the pixel still has to provide a high dynamic range
- The background light also generates electrons, which have to be stored
  - For example, the illumination units in many of today's ToF cameras can provide an illumination level of about 1 W. The sun has an illumination power of about 50 W/m^2 after the optical band-pass filter. Therefore, if the illuminated scene has a size of 1 m^2, the light from the sun is 50 times stronger than the modulated signal

ToF disadvantages

Background light

- For **non-integrating ToF** sensors that do not integrate light over time and are using near-infrared detectors to capture the short laser pulse, direct viewing of the sun is a non-issue because the **image is not integrated over time**, rather captured within a short acquisition cycle typically less than 1µs
- Such ToF sensors are used in space applications and in consideration for automotive applications

ToF disadvantages

Interference

- In certain types of ToF devices, if several time-of-flight cameras are running at the same time, the **ToF cameras may disturb each other's measurements** (though this is not true of all ToF sensors)
- There exist several possibilities for dealing with this problem
  - **Time multiplexing**: A control system starts the measurement of the individual cameras consecutively, so that only one illumination unit is active at a time
  - **Different modulation frequencies**: If the cameras modulate their light with different modulation frequencies, their light is collected in the other systems only as background illumination, but does not disturb the distance measurement

ToF disadvantages

Interference

- For direct ToF type cameras that use a **single laser pulse for illumination**, because the single laser pulse is short (e.g., 10ns), the round trip ToF to and from the objects in the field of view is correspondingly short (e.g., 100m = 660ns ToF round trip), for an imager capturing at 30Hz, the probability of an interfering interaction is the time that the camera acquisition gate is open divided by the time between laser pulses or approximately 1 in 50,000 (0.66 µs divided by 33 ms)

ToF disadvantages

Multiple reflections

- In contrast to laser scanning systems where a single point is illuminated, the time-of-flight cameras **illuminate a whole scene**
- For a phase difference device (amplitude modulated array), due to multiple reflections, the **light may reach the objects along several paths**
- Therefore, the **measured distance** may be **greater than the true distance**
- Direct ToF imagers are **vulnerable if the light is reflecting from a specular surface**

Applications

Automotive applications

- ToF cameras are used in **assistance and safety functions for advanced automotive applications** such as **active pedestrian safety**, **pre-crash detection** and **indoor applications like out-of-position** (OOP) detection

Applications

Human-machine interfaces and gaming

- As ToF cameras provide distance images in real time, it is easy to **track movements of humans**. This allows **new interactions** with consumer devices such as televisions
- Another topic is to use this type of cameras to **interact with games** on **video game consoles**. The second-generation Kinect sensor which is a standard component of the Xbox One console uses a time-of-flight camera for its range imaging, enabling natural user interfaces and gaming applications using computer vision and gesture recognition techniques

Applications

Human-machine interfaces and gaming

- Creative and Intel also provide a similar type of interactive gesture ToF camera for gaming, the ***Senz3D*** based on the ***DepthSense*** 325 camera of ***Softkinetic***
- Infineon and PMD Technologies enable tiny integrated 3D depth cameras for close-range gesture control of consumer devices like all-in-one PCs and laptops

Applications

Measurement and machine vision

- Range image with height measurements
- Other applications are measurement tasks, e.g., for the fill height in silos
- In industrial machine vision, the ToF camera helps to classify objects and help robots find the items, for instance on a conveyor
- Door controls can distinguish easily between animals and humans reaching the door

Applications

Robotics

- Another use of these cameras is the field of robotics: **Mobile robots** can build up a **map of their surroundings** very quickly, enabling them to avoid obstacles or follow a leading person. As the distance calculation is simple, only little computational power is used

Earth topography

- ToF cameras have been used to obtain **digital elevation models** of the Earth's surface topography, for studies in geomorphology

NON-CONTACT SCANNERS: TRIANGULATION BASED

Non-contact passive scanners

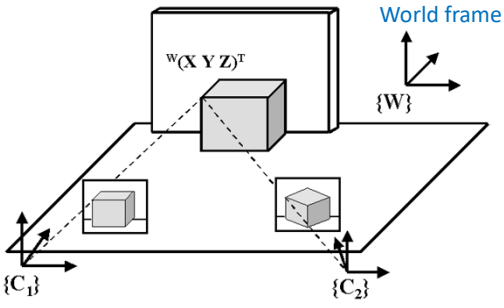
- We can now shortly introduce the **non-contact passive scanners** (see the classification on slide 17)
- **Passive 3D imaging solutions** do not emit any kind of radiation themselves, but instead rely on **detecting reflected ambient radiation**
- Most solutions of this type detect visible light because it is a readily available ambient radiation
- Other types of radiation, such as **infrared** could also be used

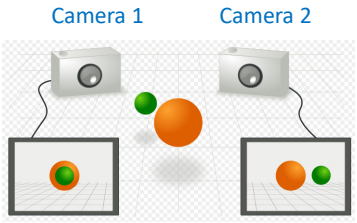
Non-contact passive scanners

- Passive methods can be **very cheap**, because in most cases they do not need particular hardware, but simple **digital cameras**
- **Stereoscopic systems** usually employ two video cameras, slightly apart, looking at the same scene
- By analyzing the slight differences between the images seen by each camera, it is possible to determine the distance at each point in the images
- This method is based on the same principles driving **human stereoscopic vision**

Passive triangulation: stereo vision

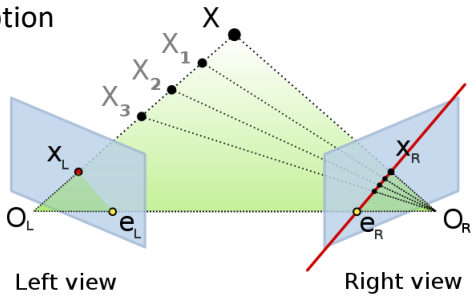
- **Correspondence problem**: same points viewed in the two views must be recognized – **matching pairs**
- Geometric constraints ⇒ search along **epipolar lines**
- 3D reconstruction of matched pairs by **triangulation**





Idea of the epipolar geometry

- When **two cameras view a 3D scene** from **two distinct positions**, there are a number of **geometric relations** between the 3D points and their projections onto the 2D images that lead to constraints between the image points
- These relations are studied by the ***epipolar geometry*** and are derived based on the assumption that the cameras can be approximated by the **pinhole camera model**
- If X_L and X_R are known**, X** can be calculated from the coordinates of the two image points

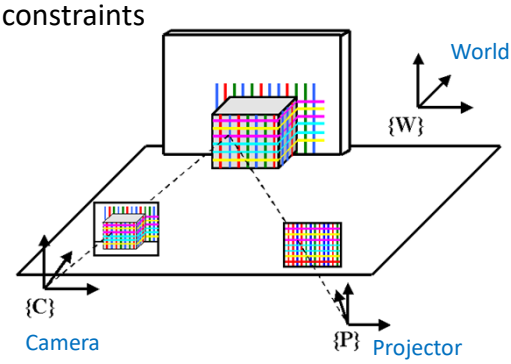


Non contact active scanners based on triangulation

- The idea of the **epipolar geometry** used by **passive stereo vision** is also used by **triangulation based 3D laser scanners**
- These are also **active scanners** that use laser light to probe the environment
- With respect to ToF 3D laser scanners, the triangulation laser **shines a laser on the object** and exploits a **camera** to look for the location of the **laser dot**
- Depending on **how far away the laser strikes a surface**, the **laser dot appears at different places in the camera's field of view**
- This technique is called ***triangulation*** because the laser dot, the camera and the laser emitter form a triangle

Active triangulation: structured light

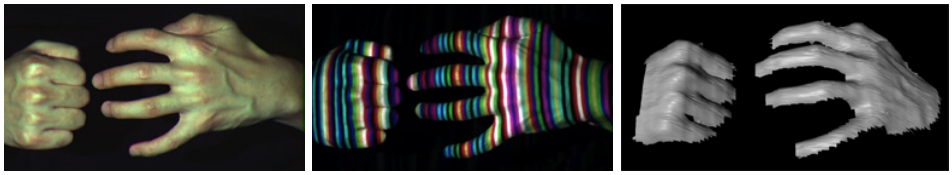
- One of the cameras is replaced by a **light emitter** (active)
- **Correspondence problem** is solved by **searching the pattern in the camera image** (pattern decoding)
- No geometric constraints



89

Structured light

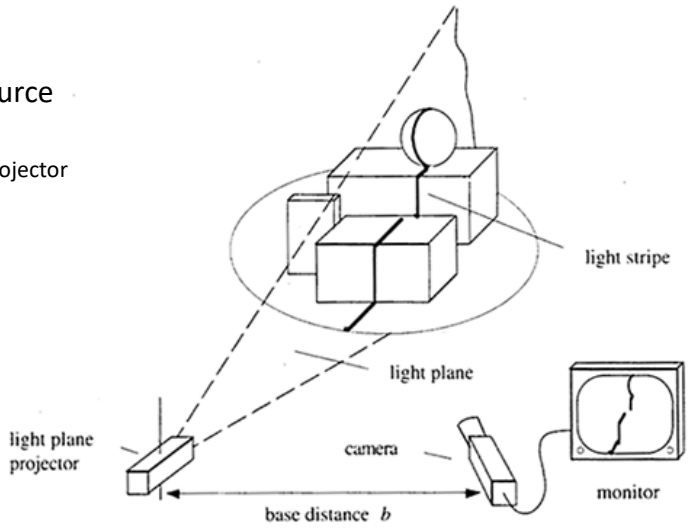
- The technique of analyzing a known pattern is called **structured light**
- Structured light general principle: **project a known pattern onto the scene** and **infer depth from the deformation of that pattern**



Computer Graphics and 3D 90

Active triangulation: general setup

- One camera
- One light source
  - Types
    - Slide projector
    - Laser
  - Projection
    - Spot
    - Stripe
    - Pattern



Structured light scanners

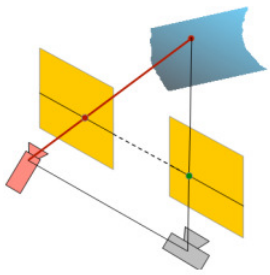
- Projecting a **narrow band of light** onto a 3D shaped surface produces a **line of illumination** that **appears distorted from other perspectives than that of the projector**, and can be used for an exact geometric reconstruction of the surface shape (**light section**)
- A faster and more versatile method is the **projection of patterns consisting of many stripes at once**, or of arbitrary fringes, as this allows for the **acquisition of a multitude of samples simultaneously**
- Seen from different viewpoints, the **pattern appears geometrically distorted** due to the surface shape of the object

Structured light scanners

- There is a continuum of methods we can use to establish **correspondences** for a structured light scanner
  - At one extreme are **single-stripe systems**. These never have the problem of ambiguity, so they produce **very high-quality data**, but they take a **long time** to do it
  - At the other extreme are systems that try to get all the data at once by projecting **lots of stripes** (or, in this case, **dots** - we will see in more details the Kinect 1 for Xbox). They are **fast**, since they **get everything with just a single frame**, but they tend to be a lot **more fragile** and get **confused by discontinuities**
  - In the middle are methods that use a **few frames** to get depth, by **flashing stripes on and off over time**, and conveying a code about which stripe is which through this pattern of on/off flashing

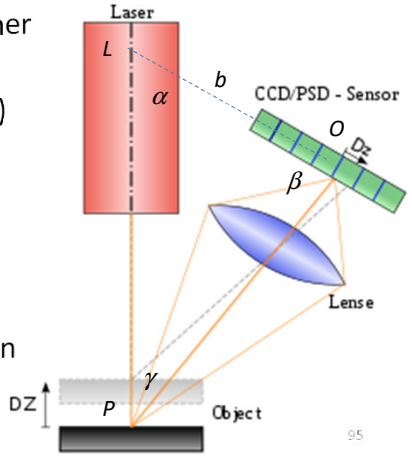
Basic principle of triangulation

- The challenge in optical triangulation is obtaining **correspondence** between **points in the projected pattern** and **pixels in the image**
- Light projection
  - If we project a single point, matching is unique
  - But many images needed to cover the object



Active triangulation

- The **length of one side of the triangle**, the **distance** between the camera and the laser emitter is known (b)
- The **angle** of the laser emitter corner is also known (α)
- The **angle** of the camera corner (β) can be determined by looking at the location of the laser dot in the camera's field of view
- These three pieces of information fully determine the shape and size of the triangle and give the location of the **laser dot corner** of the triangle

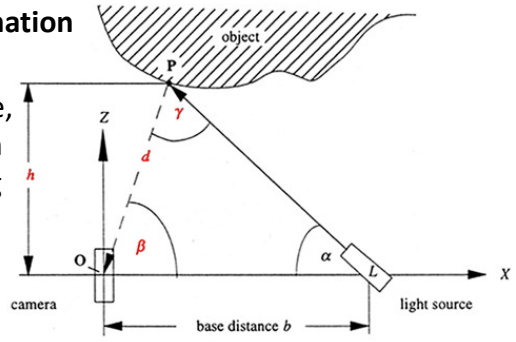


95

Active triangulation

- Assume **point-wise illumination by laser beam**, only 2D
- **O**, **L** and **P** define a triangle, and we define the position of **P** by ***triangulation*** using basic formulas about triangles such as the ***sine law***

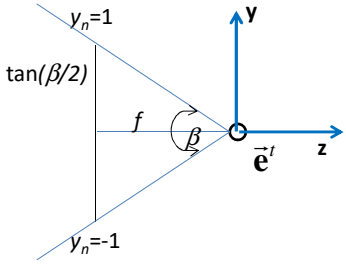
$$\frac{d}{\sin\alpha} = \frac{b}{\sin\gamma}$$



- If follows that $d = \frac{b \cdot \sin\alpha}{\sin\gamma} = \frac{b \cdot \alpha}{\sin(\pi - \alpha - \beta)} = \frac{b \cdot \alpha}{\sin(\alpha + \beta)}$

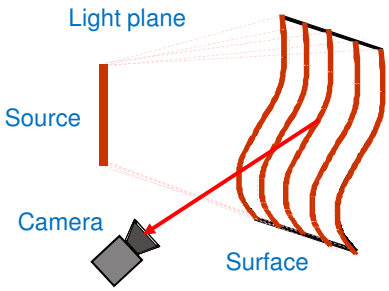
Active triangulation

- Finally, **P** = (*d* cosβ, *d* sinβ)
  - Note that β is determined by the position of the projected illuminated point **P** in the image
- In summary, known *b* and α, triangulation allow us to find the following coordinates
  - β can be found by **projection geometry**: given image coordinate and **focal length** *f* -> calculate β from *f* = tan(*β/2*)
  - given *b*, α, β -> calculate *d* = β*sin(α)/sin(α + β)
  - x_0 = d*cos(β)
  - z_0 = h = d*sin(β)

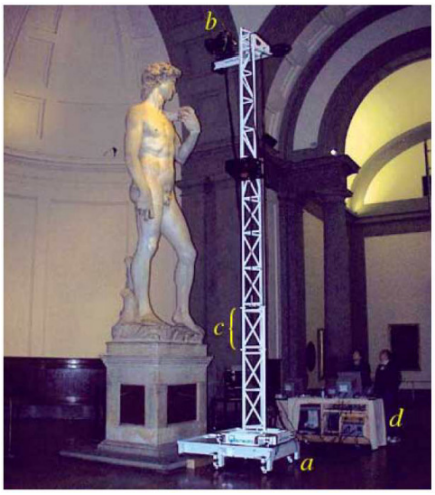


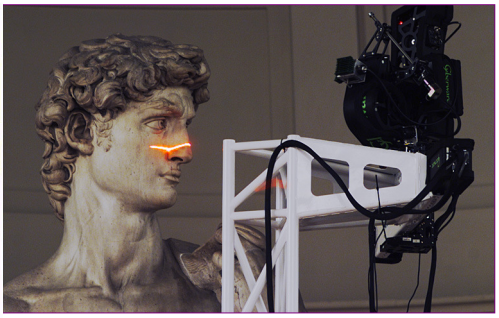
Light stripe scanning – single stripe

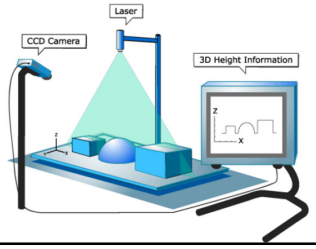
- Optical triangulation
  - Project a single stripe of laser light
  - Scan it across the surface of the object
  - This is a very precise version of structured light scanning
  - Good for high resolution 3D, but needs many images and takes time



Light stripe scanning – single stripe





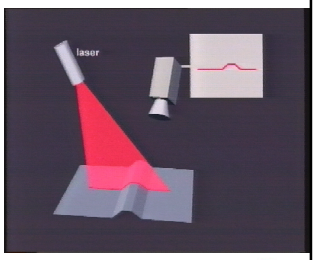


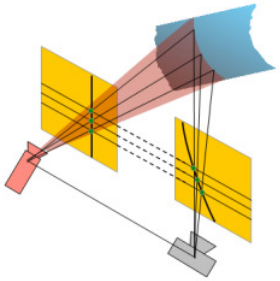
Digital Michelangelo Project
http://graphics.stanford.edu/projects/mich/

99

Light stripe scanning – single stripe

- In most cases a laser stripe, instead of a single laser dot, is swept across the object to speed up the acquisition process
  - For a calibrated projector-camera system, the epipolar geometry is known
  - Can project a **line** instead of a single point
  - A lot fewer images needed (one for all points on a line)
  - A point has to lie on the corresponding epipolar line
    - Matching is still unique
    - The depth depends only on the image point location

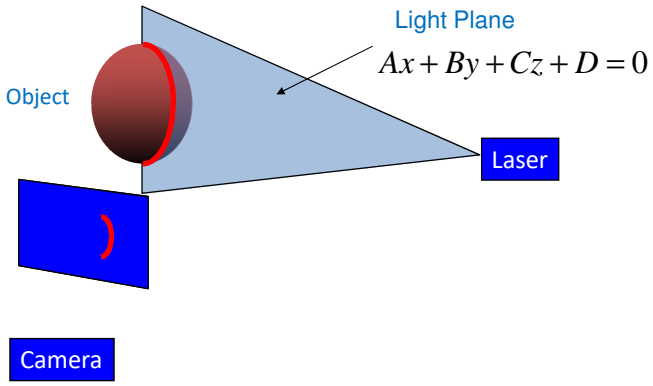




100

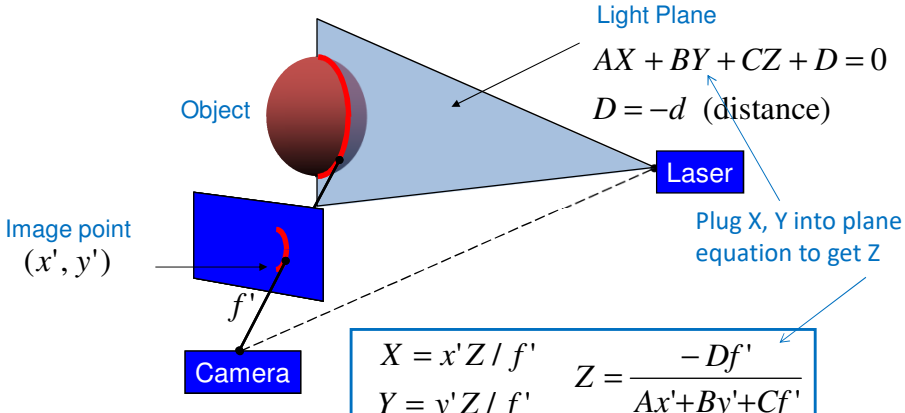
Triangulation

- Project laser stripe onto object



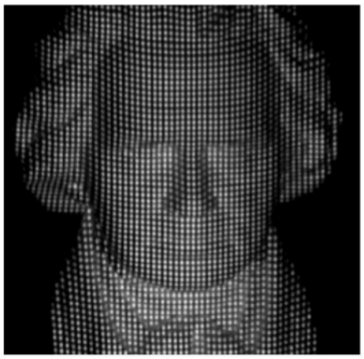
Triangulation

- Depth from ray-plane triangulation
  - Intersect camera ray with light plane



Pattern projection

- Project a **pattern** instead of a single **point** or **line**
- Needs only a single image, one-shot recording
- The correspondence between observed edges and projected image is not directly measurable
  - The matching is not unique

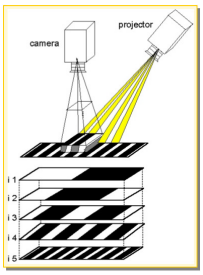


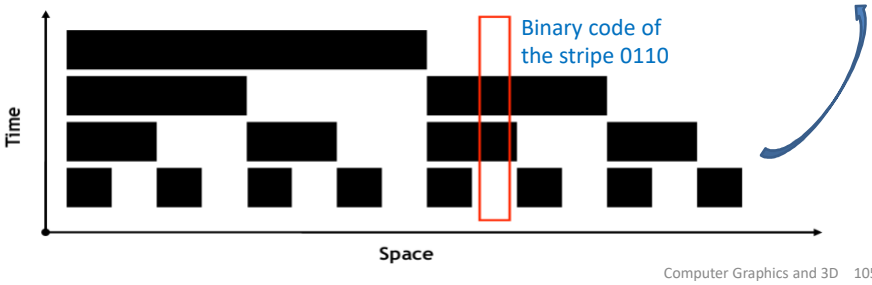
Pattern projectors: what is the problem?

- **Problem**: When multiple stripes or a grid of points are projected correspondences are not unique. Which stripe matches which?
- General strategy: use special illumination patterns to simplify matching and guarantee dense coverage
- **Structured light** also known as
  - Active stereo
  - White light scanning

Pattern projectors: multi-stripe projectors

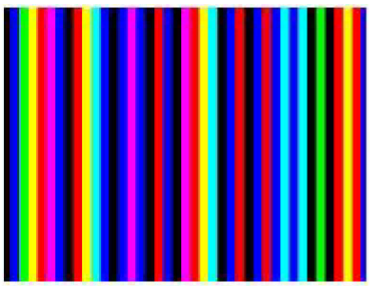
- **Time-coded light patterns**
  - "Best compromise" between single stripes and stripe pattern
  - Use a sequence of binary patterns → *log(images width)*
  - Each stripe has a **unique binary illumination code**

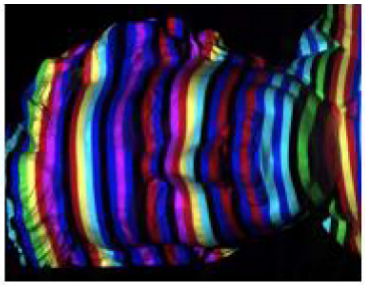




Pattern projectors: colored stripes

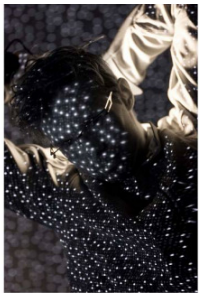
- Can be designed such that local patterns are unambiguous
- Difficult to use for colored surfaces





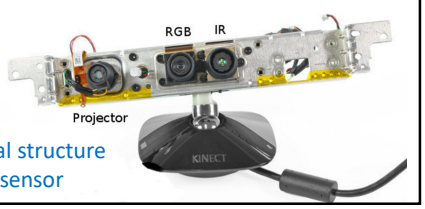
Pattern projectors: pseudo random pattern

- **Spatial neighborhood**: The **code-word** that labels each pixel is obtained from a neighborhood of pixels around it
  - This coding has to be unique per position in order to recognize each point in the pattern
  - The decoding stage is more difficult because the whole neighborhood can not always be recovered due to occlusions and shadows



Used in the ***Kinect*** 1 for Xbox 360 System

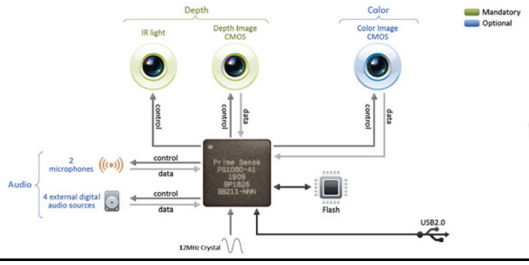
Microsoft Kinect for Xbox 360 (Kinect 1)



Internal structure of the sensor

Remarks

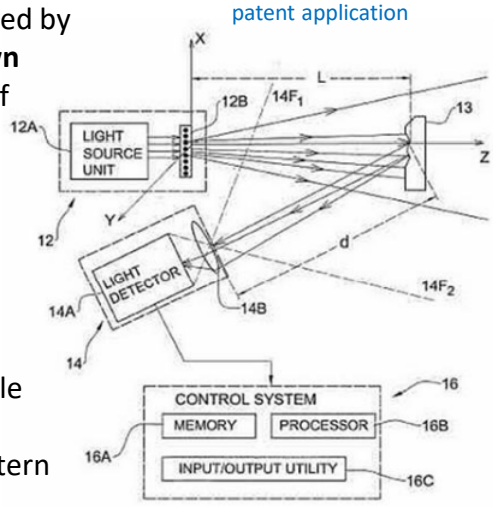
- ***Microsoft*** licensed this technology from a company called ***PrimeSense***
- The depth computation is all done by the *PrimeSense* hardware built into Kinect
- Details are not publicly available; the following description is speculative (based mostly on *PrimeSense* patent applications) and may be inaccurate or wrong



Principles of Kinect 1

- The depth map is constructed by analyzing a projected **known pattern** (***speckle pattern***) of **near-infrared laser light**
- CMOS IR camera observes the scene
- Calibration between the projector and camera has to be known
- Triangulation of each speckle between a virtual image (pattern) and observed pattern

Image from ***PrimeSense*** patent application

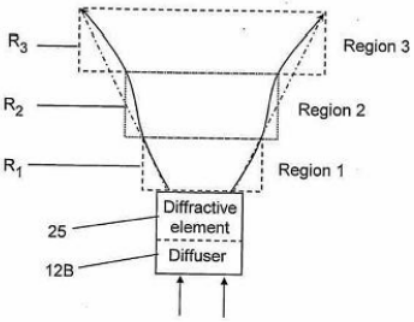


110

Principles of Kinect 1

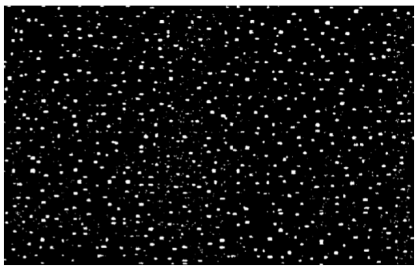
- Kinect uses 3 **different sizes of speckles** for 3 **different regions of distances**
  - **First region**: Allows us to obtain a **high accurate** depth surface for **near objects** approximately (0.4 – 1.2 m)
  - **Second region**: Allows us to obtain **medium accurate** depth surface approximately (1.2 – 2.0 m)
  - **Third region**: Allows us to obtain **a low accurate** depth surface in **far objects** approximately (2.0 – 3.5 m)

Image from ***PrimeSense*** patent application



Speckle pattern

- The Kinect uses infrared laser light, with a ***speckle pattern***
- The Kinect uses an **infrared projector** and **sensor**; it does not use its RGB camera for depth computation
- The Kinect combines structured light with two classic computer vision techniques
  - ***depth from focus***,
  - and, ***depth from stereo***



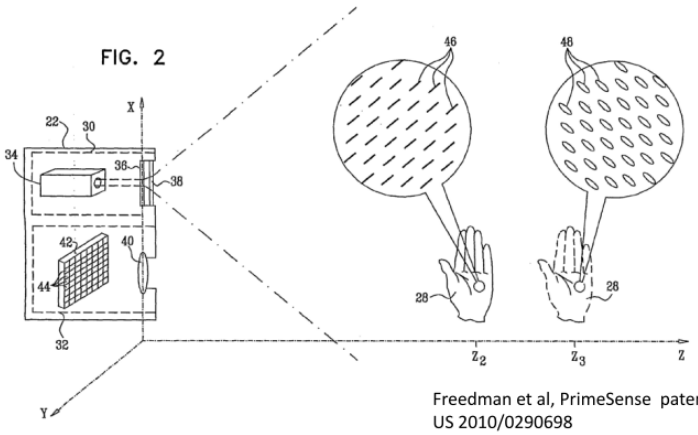
Depth from focus

- **Depth from focus** uses the principle that **stuff** that is **more blurry** is **further away**
- The Kinect dramatically improves the accuracy of traditional depth from focus
- The Kinect uses a special ("astigmatic") lens with **different focal length** in x- and y- directions
- A **projected circle** then becomes an **ellipse** whose **orientation depends on depth**

Depth from focus

- The **astigmatic lens** causes a **projected circle** to become an **ellipse** whose **orientation depends on depth**

Image from ***PrimeSense*** patent application



Freedman et al, PrimeSense patent application US 2010/0290698

Disparity

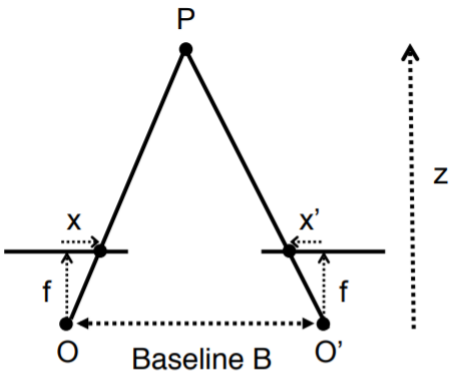
- ***Binocular disparity*** refers to the **difference in image location** of an object seen by the **left** and **right eyes**, resulting from the eyes' horizontal separation (***parallax***)
- The brain uses binocular disparity to extract **depth information** from the two-dimensional retinal images in stereopsis
- In Computer Vision, binocular disparity refers to the **difference in coordinates** of **similar features** within two **stereo images**

Disparity

- Knowledge of disparity can be used for depth/distance calculation
- **Disparity** and **distance** from the cameras are **inversely related**: as the **distance from the cameras increases**, the **disparity decreases**
- This allows for **depth perception in stereo images**
- Using geometry and algebra, the points that appear in the 2D stereo images can be mapped as coordinates in 3D space

Disparity

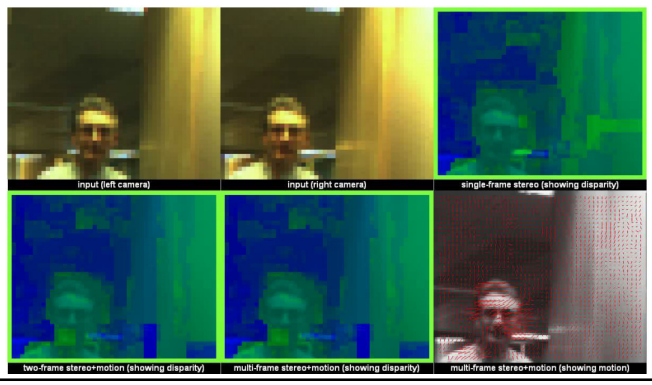
Disparity and **distance** from the cameras are **inversely related**: as the **distance from the cameras increases**, the **disparity decreases**



$$x - x' = \frac{B \cdot f}{z} = \text{disparity}$$

Depth from stereo uses *parallax*

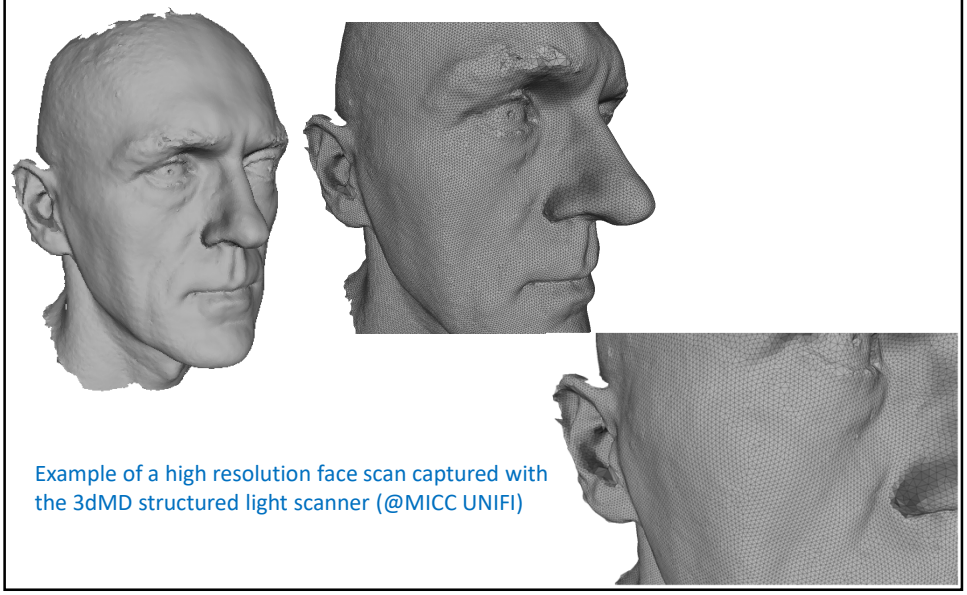
- If you look at the scene from another angle, stuff that is close gets shifted to the side more than stuff that is far away
- The Kinect analyzes the shift of the **speckle pattern** by projecting from one location and observing from another



Example: 3dMD structured light scanner

- Example of the structured light patterns used by the 3dMD scanner (@MICC UNIFI)
- This works similarly to Kinect, by projecting a pattern on the scene

Example: 3dMD structured light scanner



Example of a high resolution face scan captured with the 3dMD structured light scanner (@MICC UNIFI)

References

[1] Fausto Bernardini, Holly E. Rushmeier, "The 3D Model Acquisition Pipeline," *Comput. Graph. Forum* 21 (2): 149–172, 2002. *doi:10.1111/1467-8659.00574*

[2] Brian Curless, "From Range Scans to 3D Models," *ACM SIGGRAPH Computer Graphics* **33** (4): 38–41, 2000. doi:10.1145/345370.345399

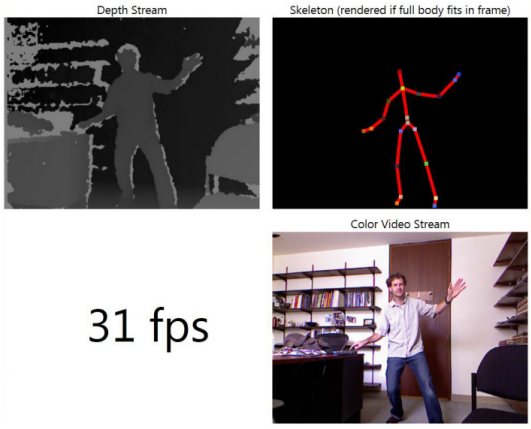
[3] S. Burak Gokturk, Hakan Yalcin and Cyrus Bamji, "A Time-Of-Flight Depth Sensor – System Description, Issues and Solutions," *IEEE Conf. on Computer Vision and Pattern Recognition* (CVPR), pages 35-45, 2004

Extracting skeleton of the human body from depth images

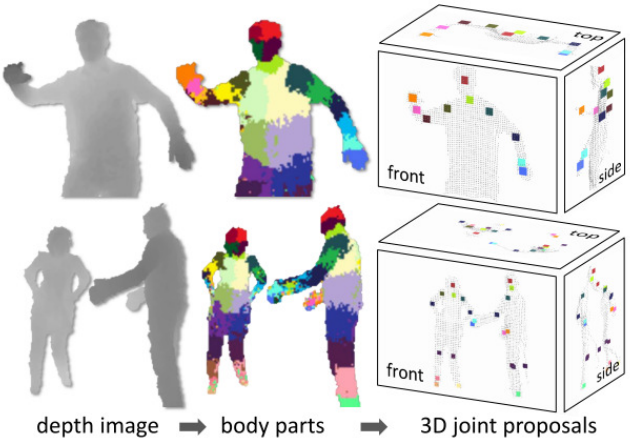
Lecture 12 – B

Overview

- The Kinect camera provides, in addition to a depth image of the scene, a body skeleton, which estimates the 3D position of the joints of the body in the case one or more persons are present in the scene
- In the following, we will investigate how this is obtained as proposed in [1]



Overview



From a single input depth image, a **per-pixel body part distribution is inferred** (colors indicate the most likely part labels at each pixel, and correspond in the joint proposals).
Local modes of this signal are estimated to give high-quality proposals for the 3D locations of body joints, even for multiple users

Depth image

- Pixels in a depth image indicate calibrated depth in the scene, rather than a measure of intensity or color
- The Kinect 1 camera gives a 640x480 image at 30 frames per second with depth resolution of a few centimeters
- Depth cameras offer several advantages over traditional intensity sensors, working in **low light levels**, giving a **calibrated scale estimate**, being color and texture invariant, and **resolving silhouette ambiguities in pose**
- They also greatly simplify the task of **background subtraction**
- Also, it is straightforward to **synthesize realistic depth images of people** and thus build a large training dataset cheaply

Body part labeling

- Several localized **body part labels** are defined that **densely cover the body** (parts are color-coded in Figures)
- Some of these parts are defined to directly localize particular skeletal joints of interest, while others fill the gaps or could be used in combination to predict other joints
- This intermediate representation transforms the problem into one that can readily be solved by efficient **classification algorithms**

Body part labeling

- The pairs of depth and body part images are used as fully labeled data for learning the classifier
- 31 body parts are used (Left, Right, Upper, loWer)
  - LU/RU/LW/RW head, neck, L/R shoulder,
  - LU/RU/LW/RW arm, L/R elbow, L/R wrist, L/R hand,
  - LU/RU/LW/RW torso,
  - LU/RU/LW/RW leg, L/R knee, L/R ankle, L/R foot

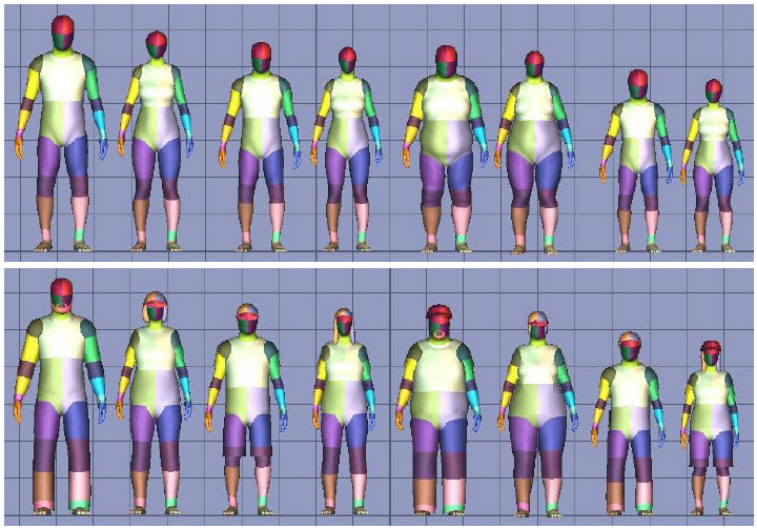
Synthetic and real data

- Starts with 100,000 depth images with **known skeletons** (from a **motion capture system**)
- For each real image, render dozens more using computer graphics techniques
- Use **computer graphics** to render all sequences for 15 **different body types**, and vary several other parameters
- Thus obtain over a **million training examples**

Synthetic and real data

Pairs of depth image and ground truth body parts.
Note wide variety in pose, shape, clothing, and crop

Synthetic data



Depth image features

- Simple **depth comparison features** are used at a given pixel $\mathbf{x}$

$$f_\theta(I, \mathbf{x}) = d_I\left(\mathbf{x} + \frac{\mathbf{u}}{d_I(\mathbf{x})}\right) - d_I\left(\mathbf{x} + \frac{\mathbf{v}}{d_I(\mathbf{x})}\right) \qquad (12.1)$$

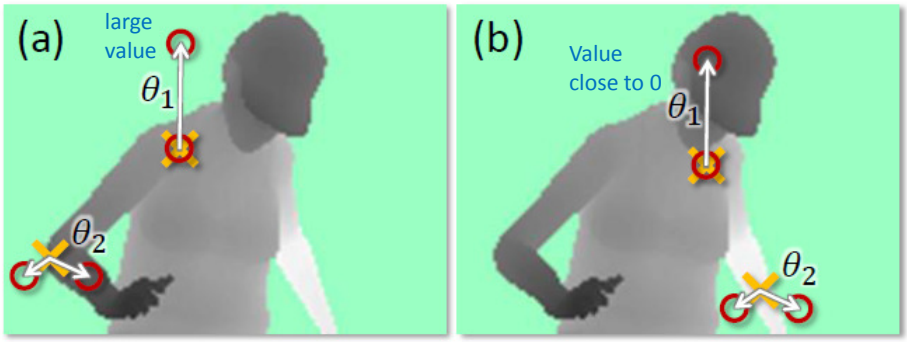
  where $d_I(\mathrm{x})$ is the **depth at pixel** $\mathbf{x}$ in image I, and parameters $\theta = (\mathrm{u}, \mathrm{v})$ describe offsets u and v

- The **normalization** of the offsets by $d_I(\mathrm{x})$ ensures the **features are depth invariant**: at a given point on the body, a fixed world space offset will result, whether the pixel is close or far from the camera
- The features are thus **3D translation invariant** (modulo perspective effects)

Depth image features

- If an offset pixel lies on the **background** or **outside** the bounds of the image, the depth probe $d_I(\mathrm{x}')$ is given a **large positive constant value**
- Next slide illustrates two features at different pixel locations x
  - Feature $f_{\theta 1}$ looks upwards: Eq.(12.1) will give a large positive response for pixels x near the top of the body, but a value close to zero for pixels x lower down the body
  - Feature $f_{\theta 2}$ may instead help find thin vertical structures such as the arm
- Individually these features provide only a **weak signal** about which part of the body the pixel belongs to, but in combination in a **decision forest** they are sufficient to accurately disambiguate all trained parts

Depth image features



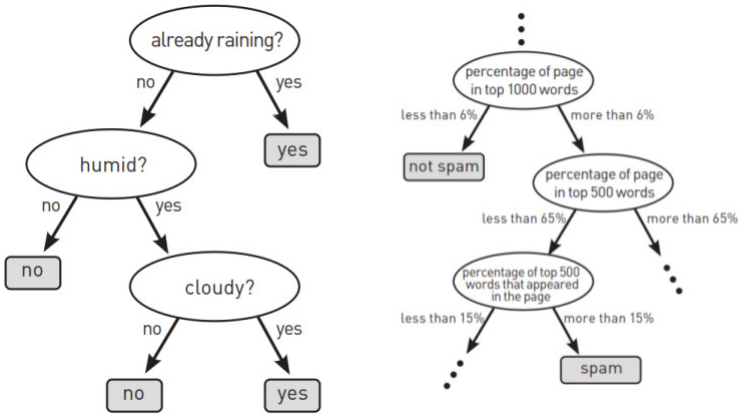
Depth image features. The yellow crosses indicate the pixels $\mathbf{x}$ being classified.
The red circles indicate the offset pixels as defined in Eq.(12.1).
In (a), the two example features give a large depth difference response.
In (b), the same two features at new image locations give a much smaller response

Depth image features

- The design of these features was motivated by their **computational efficiency**
  - No preprocessing is needed
  - Each feature needs only read at most 3 image pixels and perform at most 5 arithmetic operations (2 divisions, 2 additions, 1 subtraction)
  - The features can be straightforwardly implemented on the GPU
- Given a larger computational budget, one could employ potentially more powerful features based on, for example, depth integrals over regions, curvature, or local descriptors

Decision tree

- A ***randomized decision forest*** is a more sophisticated version of the classic ***decision tree***
- A decision tree is like a **pre-planned game** of "**questions**"



Decision tree

- To learn a decision tree, you choose as the **next question** the one that is most "**useful**" on (the relevant part of) the **training data**
  - E.g., for umbrella tree, is "raining?" or "cloudy?" more useful?
- In practice, "**useful**" = ***information gain*** G (which is derived from **entropy** H)

$$G(\phi) = H(Q) - \sum_{s \in \{\mathrm{l,r}\}} \frac{|Q_s(\phi)|}{|Q|} H(Q_s(\phi))$$

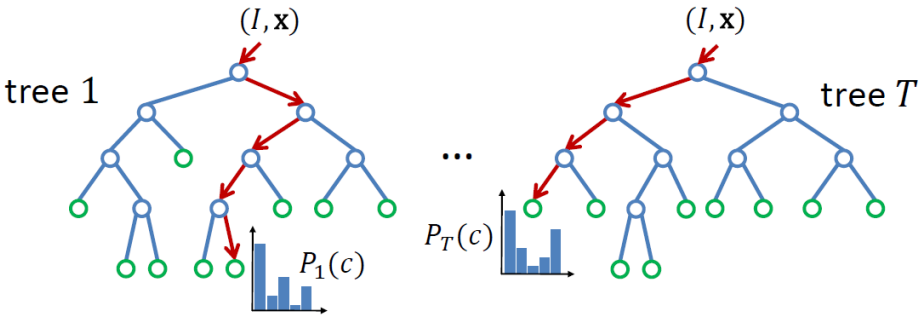
See slides 18-19 for more on the symbols used in this equation

Randomized decision forests

- A ***forest*** is an ensemble of *T* ***decision trees***, each consisting of ***split*** and ***leaf*** nodes
- Each **split** node consists of a **feature** f_θ and a **threshold** τ
- To classify pixel $\mathbf{x}$ in image I, one starts at the root and repeatedly evaluates Eq.(12.1), **branching left or right** according to the **comparison to threshold** τ
- At the leaf node reached in tree *t*, a **learned distribution** $P_t(c|I, \mathrm{x})$ over **body part labels** c is stored
- The distributions are averaged together for all trees in the forest to give the final classification

$$P(c|I, \mathbf{x}) = \frac{1}{T} \sum_{t=1}^{T} P_t(c|I, \mathbf{x}) \ .$$

Randomized decision forests



A forest is an ensemble of trees. Each tree consists of ***split nodes*** (blue) and ***leaf nodes*** (green). The red arrows indicate the different paths that might be taken by different trees for a particular input

Training

- Each tree is trained on a **different set of randomly synthesized images**
- A random subset of 2000 example pixels from each image is chosen to ensure a roughly even distribution across body parts
- Each tree is **trained** using the following algorithm
  1. Randomly propose a set of **splitting candidates** $\phi = (\theta, \tau)$ (**feature parameters** θ and **thresholds** τ)
  2. Partition the set of examples $Q = \{(I, x)\}$ into **left** and **right subsets** by each ϕ

$$Q_l(\phi) = \{ (I, \mathbf{x}) \mid f_\theta(I, \mathbf{x}) < \tau \}$$
$$Q_r(\phi) = Q \setminus Q_l(\phi)$$

  3. ...

Training

3. Compute the ϕ giving the largest **gain in information**

$$\phi^\star = \operatorname*{argmax}_\phi G(\phi)$$

$$G(\phi) = H(Q) - \sum_{s\in\{\mathrm{l,r}\}} \frac{|Q_s(\phi)|}{|Q|} H(Q_s(\phi))$$

There is an information gain if the **entropy after the split** is **lower** than the **entropy before the split**

where ***Shannon*** **entropy** $H(Q)$ is computed on the **normalized histogram of body part labels** $l_I(\mathrm{x})$ for all $(\mathrm{I}, \mathrm{x}) \in \mathrm{Q}$

4. If the largest gain $G(\phi^*)$ is **sufficient**, and the **depth in the tree is below a maximum**, then recurs from point 1. for left and right subsets $\mathrm{Q_l}(\phi^*)$ and $\mathrm{Q_r}(\phi^*)$

- Training 3 trees to depth 20 from 1 million images takes about a day on a 1000 core cluster

Joint position proposals

- Body part recognition as described above **infers per-pixel information**
- This information must now be **pooled across pixels** to generate **reliable proposals for the positions of 3D skeletal joints**
- These proposals are the final output of the algorithm, and could be used by a tracking algorithm to self-initialize and recover from failure
- A **local mode-finding** approach based on ***mean shift*** with a ***weighted Gaussian kernel*** is used

Joint position proposals

- A **density estimator per body part** is defined as

$$f_c(\hat{\mathbf{x}}) \propto \sum_{i=1}^{N} w_{ic} \exp\left(-\left\|\frac{\hat{\mathbf{x}} - \hat{\mathbf{x}}_i}{b_c}\right\|^2\right)$$

  where $\hat{\mathrm{x}}$ is a coordinate in 3D world space, N is the number of image pixels, w_{ic} is a pixel weighting, $\hat{\mathrm{x}}_\mathrm{i}$ is the re-projection of image pixel x_i into world space given depth $d_I(\mathrm{x}_\mathrm{i})$, and b_c is a learned per-part bandwidth
- The pixel weighting w_{ic} considers both the **inferred body part probability** at the pixel and the **world surface area of the pixel**

$$w_{ic} = P(c|I, \mathbf{x}_i) \cdot d_I(\mathbf{x}_i)^2$$

Joint position proposals

- This ensures density estimates are depth invariant and gave a small, but significant improvement in joint prediction accuracy
- Depending on the definition of body parts, the posterior $P(c|I, \mathrm{x})$ can be pre-accumulated over a small set of parts
  - For example, in the experiments the four body parts covering the head are merged to localize the head joint
- ***Mean shift*** is used to **find modes** in this **density efficiently**
  - All pixels above a learned probability threshold λ_c are used as **starting points** for part *c*
  - A **final confidence estimate** is given as a **sum of the pixel weights reaching each mode**. This proved more reliable than taking the modal density estimate

Joint position proposals

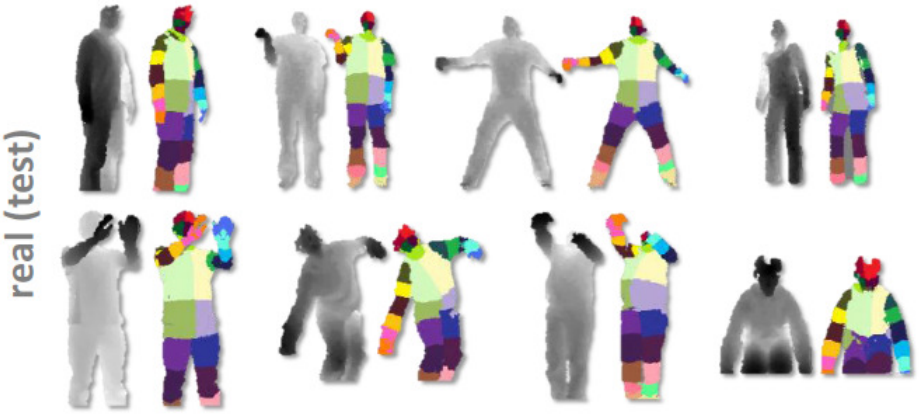
- The **detected modes** lie on the **surface** of the **body**
- Each mode is therefore pushed back into the scene by a learned *z*-offset ζ_c to produce a **final joint position proposal**
- This simple, efficient approach works well in practice
- The band-widths b_c, probability threshold λ_c, and surface-to-interior *z*-offset ζ_c are optimized per-part on a hold-out validation set of 5000 images by grid search
  - As an indication, this resulted in mean bandwidth 0.065m, probability threshold 0.14, and z-offset 0.039m

Experiments

Setting

- 3 trees, 20 deep
- 300k training images per tree
- 2000 training example pixels per image
- 2000 candidate features θ
- 50 candidate thresholds τ per feature

Real test



Example inferences

Synthetic (top row); **real** (middle); **failure modes** (bottom). Left column: **ground truth** for a neutral pose as a reference.
In each example, we see the depth image, the inferred most likely body part labels, and the joint proposals shown as front, right, and top views (overlaid on a depth point cloud). Only the most confident proposal for each joint above a fixed, shared threshold is shown

Idea for a course project

- Acquire depth sequences with Kinect, and use the skeleton to animate an avatar
  - Use together 3D acquisition, skeleton extraction using machine learning techniques, and computer graphics for visualization and rendering
- OpenNI SDK for capturing Kinect data
- Qt for windows management
- OpenGL for the 3D virtual world and avatar animation

References

[1] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman and Andrew Blake, "Real-Time Human Pose Recognition in Parts from Single Depth Images," *IEEE Conf. on Computer Vision and Pattern Recognition* (CVPR), pages 1-8, 2011

Point Cloud Triangulation

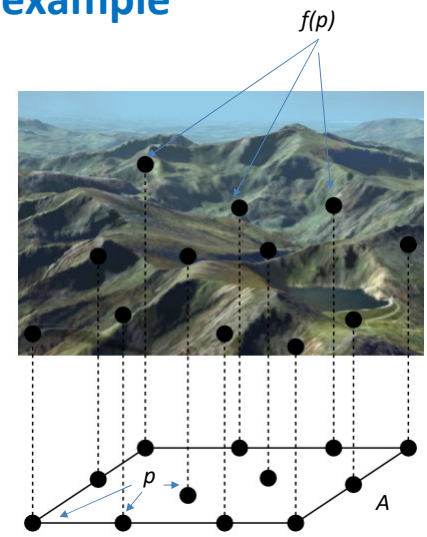
Lecture 14 – A

Motivation

- In lecture 12-A, we have seen as 3D acquisition devices capture scans in the form of clouds of 3D points or depth images
- In lecture 13-A and B, we have seen as multiple scans can be registered together in a unique model
- However, a triangulated mesh is typically needed in order to perform subsequent processing on the model surface
- In the following, we investigate how it is possible to pass from a point cloud to a triangulated mesh

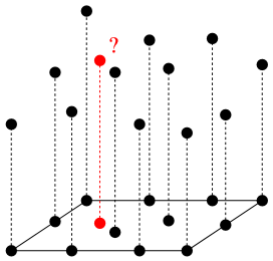
Motivating example

- A **terrain** is the graph of a function $f: A \subset R^2 \rightarrow R$
- To build a model of the terrain surface, we can start with a number of sample points where we know the height
- Height *f(p)* defined at each point *p* in *A*

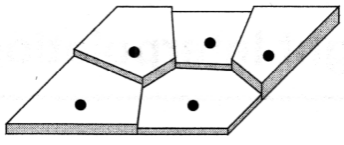


Motivating example

- How can we interpolate the height at other points?
  - *Nearest neighbor interpolation*
  - *Moving windows interpolation*
  - *Natural neighbor interpolation*
  - *Piecewise linear interpolation by a* ***triangulation***

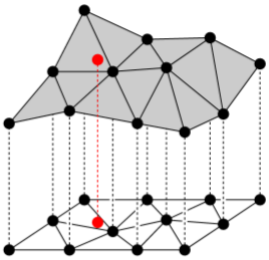


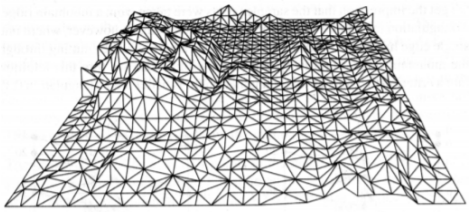
- **Example**
  Let *f(p) = height of nearest point* for points not in A
  - Does not look natural



Triangulation

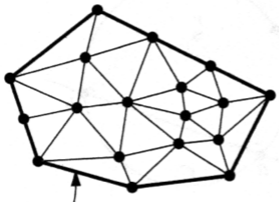
- We can use a ***triangulation***
- Let $P = \{p_1,...,p_n\}$ be a point set, a ***triangulation*** of P is a ***maximal planar subdivision*** with vertex set P
- A **maximal planar subdivision** is a subdivision S such that no edge connecting two vertices can be added to S without destroying its planarity

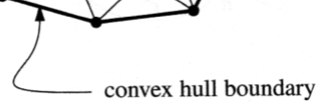




Triangulation

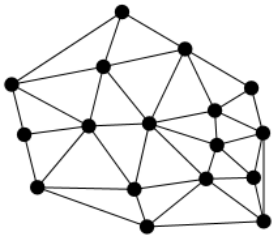
- Outer polygon must be ***convex hull***
- Internal faces must be triangles, otherwise they could be triangulated further





- Complexity
  - *2n−2−k* triangles
  - *3n−3−k* edges

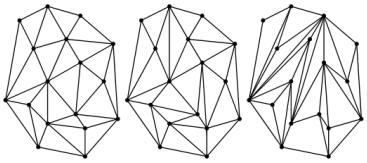
  where *k* is the number of points in *P* on the **convex hull** of *P*



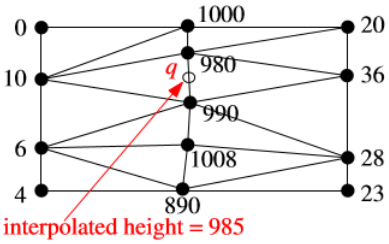
n=17
k=9
2n-2-k = 23 triangles
3n-3-k = 39 edges

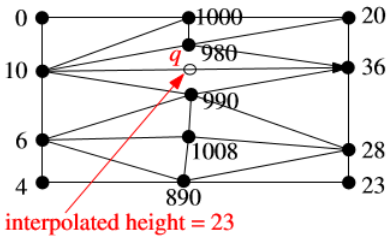
Which triangulation?

- Some triangulations are "better" than others



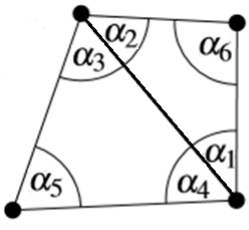
- Avoid skinny triangles, i.e., **maximize** the **minimum angle of triangulation**





Angle vector of a triangulation

- Let T be a triangulation of *P* with *m* triangles and 3*m* vertices
- Its ***angle vector*** is $A(T)=(\alpha_1,\ldots,\alpha_{3m})$, where $\alpha_1,\ldots,\alpha_{3m}$ are the angles of T sorted by **increasing value** (α_1 smallest value)



$$A(\mathcal{T}) = (\alpha_1, \ldots, \alpha_6)$$

Angle vector of a triangulation

- Let T' be another triangulation of P
- We define $A(T) > A(T')$ if $A(T)$ is ***lexicographically larger*** than $A(T')$, i.e., iff there exists an i such that $\alpha_j = \alpha'_j$ for all $j < i$ and $\alpha_i > \alpha'_i$

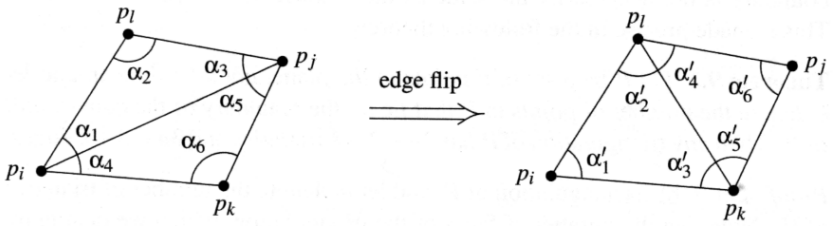
- Given two partially ordered sets A and B, the ***lexicographical order*** on the Cartesian product $A \times B$ is defined as $(a,b) \leq (a',b')$ iff $a < a'$ or $(a = a'$ and $b \leq b')$

Angle optimal triangulation

- T is ***angle optimal*** if $A(T) \geq A(T')$ for all triangulations T' of P
- **Best triangulation** is triangulation that is *angle optimal,* i.e., has the ***largest angle vector***
  - Maximizes minimum angle

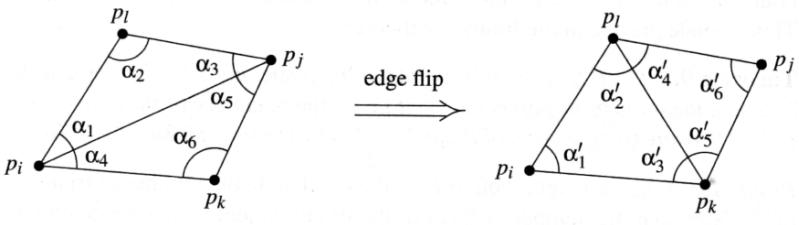
Angle optimal triangulation

- Consider two **adjacent triangles** of T (i.e., they share an edge *e*)
- If the two triangles form a convex quadrilateral, we could have an alternative triangulation by performing an ***edge flip*** on their shared edge



Illegal edges

- Edge *e* is ***illegal*** if $\min_{1 \le i \le 6} \alpha_i < \min_{1 \le i \le 6} \alpha_i'$, i.e., **flipping the edge increases the angle vector** of the triangulation
- The only difference between T containing *e,* and T' with *e* **flipped** is given by the six angles of the quadrilateral



Illegal triangulation

- If a triangulation T contains an **illegal edge** *e*, we can make *A*(T) (i.e., its **angle vector**) larger by flipping *e*
- In this case, T is an ***illegal triangulation***

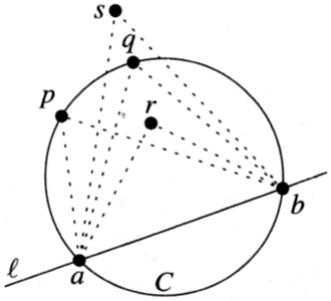
Thales's theorem

- We can use ***Thales's theorem*** to test if an edge is ***legal*** without calculating angles

Let *C* be a circle, *l* a line intersecting *C* in points *a* and *b* and *p*, *q*, *r*, and *s* points lying on the same side of *l*

Suppose that *p* and *q* lie on *C*, that *r* lies inside *C*, and that *s* lies outside *C*. Then

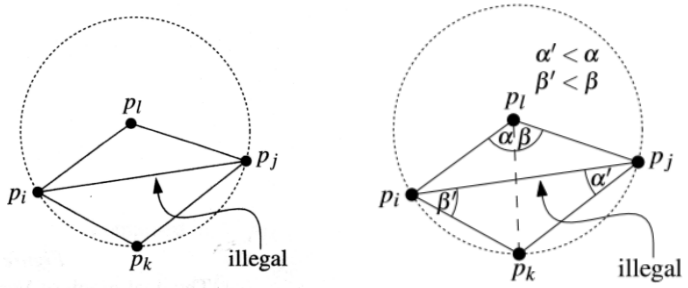
$$\measuredangle arb > \measuredangle apb = \measuredangle aqb > \measuredangle asb.$$



where $\measuredangle axb$ denotes the **angle** at vertex x of the triangle defined by the three points *a, x, c*

Testing for illegal edges

- If p_i, p_j, p_k, p_l form a ***convex quadrilateral*** and do not lie on a common circle C, exactly one of p_ip_j and p_kp_l is an **illegal edge**
- The edge p_ip_j is **illegal** iff p_l lies inside C
  - Proved using Thales's Theorem, e.g., the angle p_i -p_j -p_k is smaller than the angle p_i -p_l -p_k



15

Legal triangulation

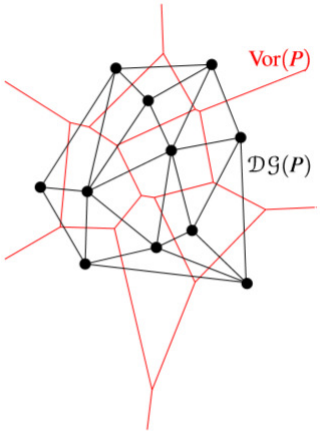
- A ***legal triangulation*** is a triangulation that does not contain any **illegal edge**

```
Algorithm LegalTriangulation(T)
Input: A triangulation T of a point set P
Output: A legal triangulation of P
1.  while T contains an illegal edge pipj
2.     do ( flip pipj )
3.       Let pipjpk and pipjpl be the two triangles adjacent to pipj
4.       Remove pipj from T, and add pkpl instead
5.  return T
```

- Algorithm terminates because there is a finite number of triangulations. **Too slow to be interesting…**

Voronoi diagram and *Delaunay* graph

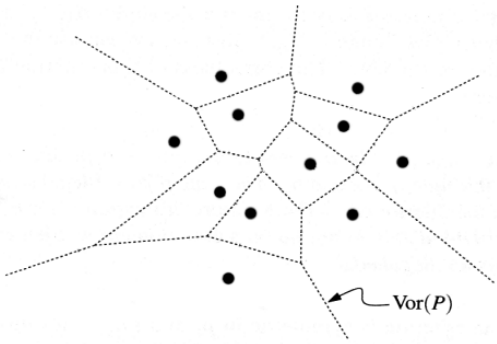
- Let P be a set of n points in the plane
- The **Voronoi* diagram** Vor(P) is the subdivision of the plane into *Voronoi* cells (sites) V(p) for all $p \in P$
- Let G be the ***dual graph*** of Vor(P)
  - The **dual graph** of a plane graph G' is a graph that has a vertex for each face of G'
- The ***Delaunay** graph* $\mathcal{DG}(P)$ is the straight line embedding of G



*Georgy Feodosevich Voronoy
Boris Nikolaevič Delone, also transliterated as **Delaunay**

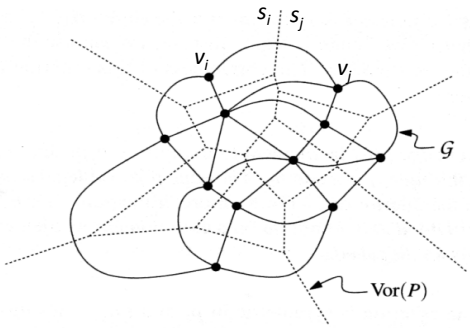
Constructing Delaunay Graph

- **Question**: How can we compute the *Delaunay graph* $\mathcal{DG}(P)$?
  - Calculate $\mathcal{Vor}(P)$
  - Place one vertex in each site of the $\mathcal{Vor}(P)$



Constructing Delaunay Graph

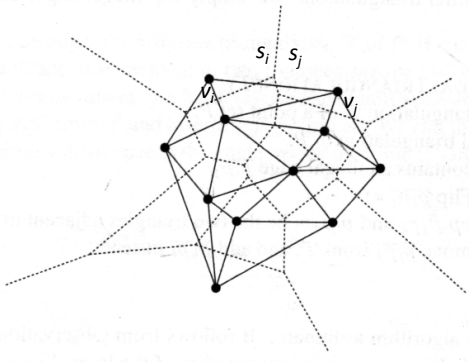
– If two sites s_i and s_j share an edge (s_i and s_j are adjacent), create an arc between v_i and v_j, the vertices located in sites s_i and s_j



Constructing Delaunay Graph

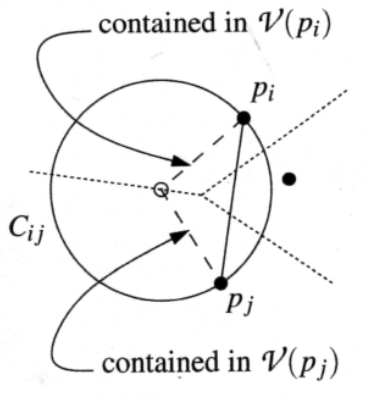
– Finally, straighten the arcs into line segments

- The resultant graph is $\mathcal{DG}(P)$



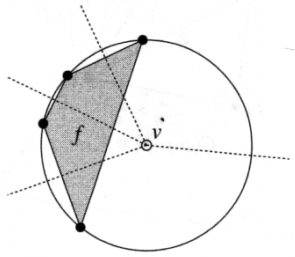
Properties of Delaunay Graph

- **Theorem**: The *Delaunay graph* $\mathcal{DG}(P)$ of a **planar point set** P is a ***plane graph***
  - **Planar graph**: A graph that can be embedded in the plane, i.e., it can be drawn on the plane in such a way that its edges intersect only at their endpoints
  - In other words, it can be drawn in such a way that **no edges cross each other**
  - **Largest empty circle property**



Delaunay triangulation

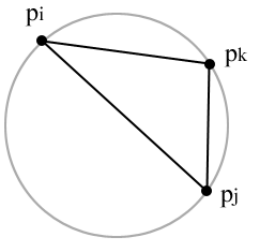
- Some sets of more than 3 points of *Delaunay graph* may lie on the same circle
- These points form *empty convex polygons*, which can be **triangulated**
- ***Delaunay Triangulation*** is a triangulation obtained by **adding zero or more edges** to the ***Delaunay Graph***



Properties of Delaunay triangles

- From the properties of *Voronoi diagrams*

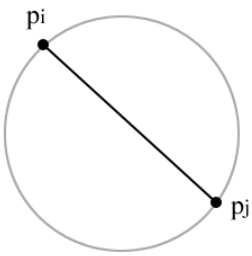
Theorem (**empty circle property**)
Three points $p_i, p_j, p_k \in P$ are **vertices** of the same **face** of the $\mathcal{DG}(P)$ **iff** the circle through p_i, p_j, p_k contains no point of P on its interior



Properties of Delaunay triangles

- From the properties of *Voronoi diagrams*

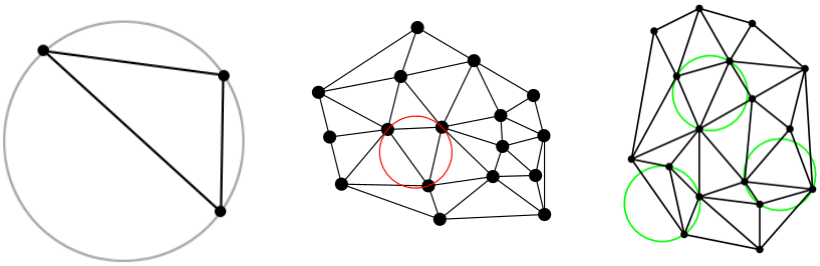
Theorem (**closest pair property**)
Two points $p_i, p_j \in P$ form an **edge** of $\mathcal{DG}(P)$ **iff** there is a closed disc C that contains p_i and p_j on its boundary and does not contain any other point of P



Delaunay triangulation

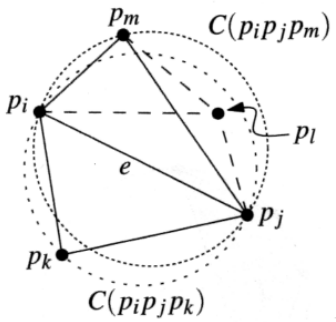
- From the previous two properties

Theorem
Let *P* be a set of points in the plane, and let *T* be a triangulation of *P*. Then *T* is ***Delaunay triangulation*** $\mathcal{DT}(P)$ of *P* **iff** the circumcircle of any triangle of *T* does not contain a point of P in its interior



Legal triangulation, revisited

- A ***triangulation*** *T* of *P* is legal **iff** *T* is a $\mathcal{DT}(P)$
- $\mathcal{DT} \rightarrow$ Legal: Empty circle property and *Thales*'s theorem implies that all $\mathcal{DT}$ are legal
- Legal $\rightarrow \mathcal{DT}$: can be proved from the definitions and via contradiction



Delaunay triangulation and angle optimal

- Let P be a set of points in the plane
- The **angle optimal triangulation** is a $\mathcal{DT}$
  - If P is in general position, $\mathcal{DT}(P)$ is **unique** and thus, is **angle optimal**
- What if multiple $\mathcal{DT}$ exist for P?
  - Not all $\mathcal{DT}$ are **angle optimal**
  - By *Thales*'s theorem, the **minimum angle** of **each** of the $\mathcal{DT}$ **is the same**
  - Thus, all the $\mathcal{DT}$ are equally "good" for the terrain problem. All $\mathcal{DT}$ **maximize the minimum angle**
- Therefore, the problem of finding a triangulation that **maximizes the minimum angle** is reduced to the problem of finding a ***Delaunay Triangulation***

How do we compute $\mathcal{DT}(P)$?

- How do we find the *Delaunay Triangulation*? There are several ways
  - By **iterative flipping from any triangulation** (see algorithm at slide 16)
  - By plane sweep
  - By **randomized incremental construction**
  - By conversion from the Voronoi diagram (we could compute $\mathcal{Vor}(P)$ then dualize into $\mathcal{DT}(P)$)
- The last three run in $O(n \log n)$ time [expected] for n points in the plane
- We sketch a $\mathcal{DT}(P)$ construction using a ***randomized incremental*** method

Randomized incremental construction

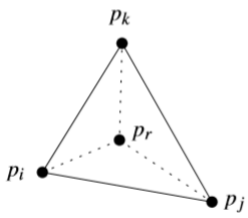
Algorithm: DelaunayTriangulation(P)
Input: A set *P* of *n+1* points in the plane
Output: A *Delaunay triangulation* of *P*

1. Initialize triangulation *T* with a **"big enough"** helper **bounding triangle** that **contains all points** *P*
2. **Randomly choose** a point p_r from *P*
3. **Find the triangle** Δ that p_r lies in // cost log(n)
4. **Subdivide** Δ into **smaller triangles** that have p_r as a vertex
5. **Flip edges** until **all edges are legal**
6. Repeat steps 2-5 until all points have been added to *T*

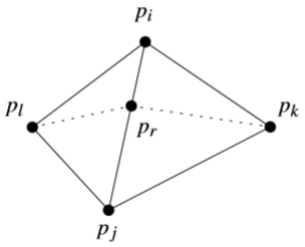
Randomized incremental construction

- The point p_r (selected at **step 2**), can lie in the interior of a triangle or fall on an edge (**step 3**)

p_r lies in the interior of a triangle



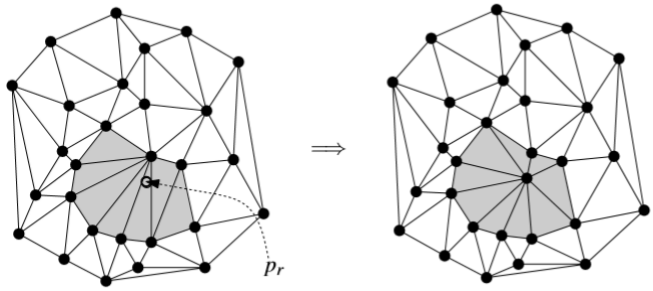
p_r falls on an edge



p_r creates new edges at step 4

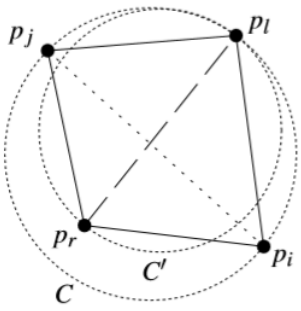
Randomized incremental construction

- At **step 4**, all edges created are incident to p_r



Randomized incremental construction

- At **step** 5, check for the correctness: for any new edge there is an empty circle through endpoints
  - New edges are legal



Cost analysis

- The Delaunay triangulation is a versatile structure that helps in constructing various things
  - *Euclidean Minimum Spanning Trees*
  - Approximations to the *Euclidean Traveling Salesperson problem*
  - *α-Hulls*
- Expected running time of algorithm is *O(n log n)* for *n* point of the plane
- Expected storage required is *O(n)*

Summary

- The *Delaunay triangulation* algorithm gives us a method to pass from a point set acquired by a 3D scanner to a set of triangles
- This triangulation represents the 3D **mesh surface** corresponding to the point set

References

[1] Steven J. Gortler, “Foundations of 3D Computer Graphics,” The MIT Press, 2012

[2] John F. Hughes et al., “Computer Graphics, Principles and Practice,” Wiley and Sons, Third Edition, 2014

Geometric Modeling

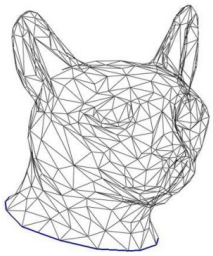
Lecture 14 – B

Geometric modeling

- In 3D shape analysis and in CG, we need concrete ways to represent shapes
- **Geometric modeling** is the topic of how to represent, create and modify such shapes
- Geometric modeling is a large topic that deserves (and has) its own books (for example [3, 4, 5])
- In the following, we address the following topics
  - Mesh representation
  - Implicit surfaces
  - Parametric patches
  - Subdivision surfaces

Triangle soup

- For rendering in OpenGL, the most obvious representation is ***triangle soup***
  - a set of triangles, each described by three vertices
- These data can often be better organized to reduce redundancy
- For example, in many cases, it makes sense to store each vertex only once, even when it is shared by many triangles
- Additionally, the connectivity information can be represented more compactly using representations such as "***triangle fans***" and "***triangle strips***" [6]



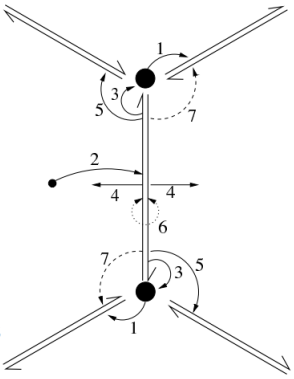
A cat head is described by a soup of triangles

Meshes

- In the soup representation, there is no way to "**walk along**" the geometry
- One cannot easily (in constant time) figure out which triangles meet at an edge, or which triangles surround a vertex
- Walking along a mesh can be useful, for example, if one is trying to smooth the geometry, or if one wishes to simulate some physical process over the geometry

Meshes

- A ***mesh*** data structure is a representation that organizes the vertex, edge, and face data so that these queries can be done easily
- There are a variety of different mesh data structures and they can be tricky to implement
- For a good reference on mesh data structures see [7]



A mesh data structure, visualized here, keeps track of how the various vertices, edges, and faces fit together in the geometry

Polygon mesh representation

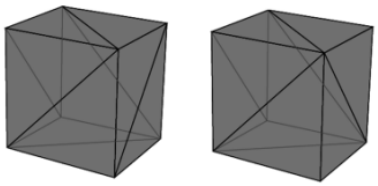
- Which representation is good?
  - Often triangles / quads only
- Compact
- Efficient for rendering
  - Fast enumeration of all faces
- Efficient for geometry algorithms
  - Finding adjacency (what is close to what)

Vertices, edges, faces

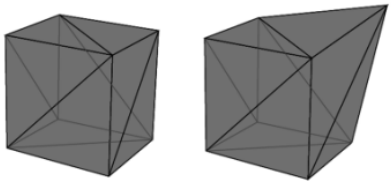
- Fundamental entities
  - n_v *vertices*
  - n_e *edges*
  - n_f *faces*
  - **Simple closed surface** $n_v - n_e + n_f = 2$
- Fundamental properties
  - **Topology**: how faces are connected
  - **Geometry**: where faces are in space
- Algorithms mostly care about **topology**

Topology vs. geometry

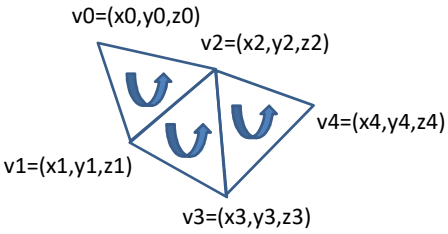
Same geometry, different topology



Same topology, different geometry



Triangles



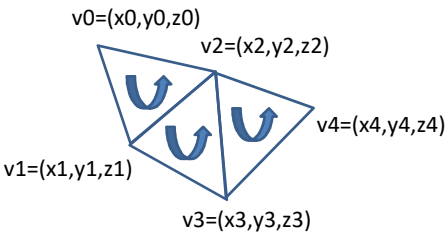
triangle[n_f][3]

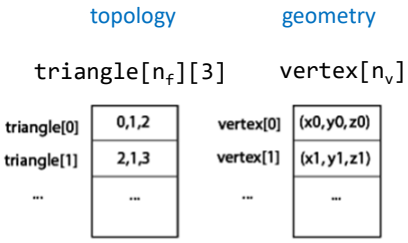
triangle[0]	**(x0,y0,z0)**	**(x1,y1,z1)**	**(x2,y2,z2)**
triangle[1]	**(x2,y2,z2)**	**(x1,y1,z1)**	**(x3,y3,z3)**
…	**…**	**…**	**…**

Triangles

- Array of vertex data
  - Data type: `triangle[`n_f`][3]`
  - Vertex stores position and optional data (normal, color)
  - 72 bytes per triangle with vertex position only (double type)
    (8 bytes * 3 vertices * 3 coordinates per vertex = 72 bytes)
- Redundant
- Adjacency is not well defined
  - Floating point errors in comparing vertices

Indexed triangles



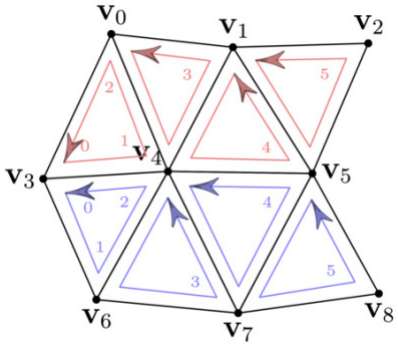


Indexed triangles

- Array of vertex data
  - Data type `vertex[`n_v`]`
  - 24 bytes per vertex with position only (8 bytes * 3 coordinates per vertex = 24 bytes)
- Array of vertex indices (3 per triangle)
  - Data type: `int triangle[`n_f`][3]`, often flattened in a single array
  - 12 bytes per triangle (4 bytes * 3 vertices = 12 bytes)
- **Topology/geometry stored separately/explicitly**
  - Adjacency queries are well defined

Triangle strips

- For many surfaces and shapes, you need to draw several connected triangles
- Since triangles share edges, reuse vertices in index list
- Requires multiple strips for general case



`strip[n_f][varyingLength]`

strip[0]	3,4,0,1,5,2
strip[1]	3,6,4,7,5,8
…	…
…	…

`vertex[n_v]`

vertex[0]	(x0,y0,z0)
vertex[1]	(x1,y1,z1)
…	…
…	…

Triangle strips

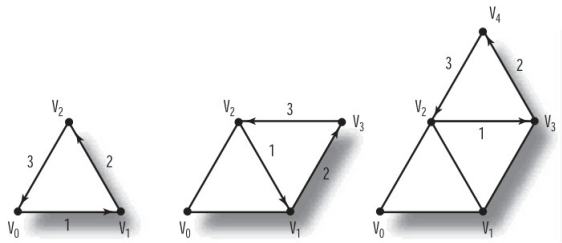
- Array of vertex data
  - `vertex[n_v]`
  - 24 bytes per vertex with position only (8 bytes * 3 coordinates per vertex = 24 bytes)
- Array of lists of vertex indices
  - `int strip[n_f][varyingLength]`
- For long lists of indices the memory is reduced to about 1/3 (i.e., a new triangle is obtained by just adding a vertex)

Triangle strips

- In OpenGL you can save a lot of time by drawing a strip of connected triangles with the `GL_TRIANGLE_STRIP` primitive

```
glDrawElements(GL_TRIANGLE_STRIP,...)
```

- Figure shows the progression of a strip of three triangles specified by a set of five vertices numbered V_0 through V_4



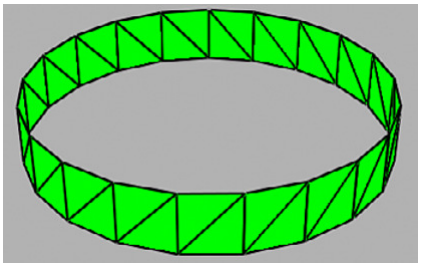
Triangle strips

- Here, you see that the vertices are **not necessarily traversed** in the **same order in which they were specified**
- The reason for this is to preserve the **winding** (**counterclockwise**) of each triangle
- The pattern is V_0, V_1, V_2; then V_2, V_1, V_3; then V_2, V_3, V_4; and so on
- There are two advantages to using a strip of triangles instead of specifying each triangle separately
  - First, after specifying the first three vertices for the initial triangle, you need to specify only a single point for each additional triangle. This saves a lot of program or data storage space when you have many triangles to draw

Triangle strips

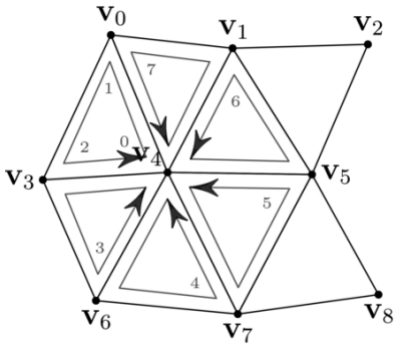
- The second advantage is mathematical performance and bandwidth savings. Fewer vertices means a faster transfer from your computer's memory to your graphics card and fewer times your vertex shader must be executed

- Here, a triangle strip is drawn to create a circular band



Triangle fans

- Similar to triangle strips, but different arrangement
- Requires many fans, so used little



fan[0]	4,0,3,6,7,5,1
fan[1]	...
...	...
...	...

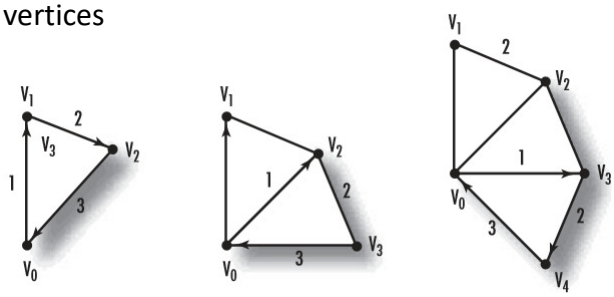
vertex[0]	(x0,y0,z0)
vertex[1]	(x1,y1,z1)
...	...
...	...

Triangle fans

- You can use `GL_TRIANGLE_FAN` to produce a group of connected triangles that fan around a central point

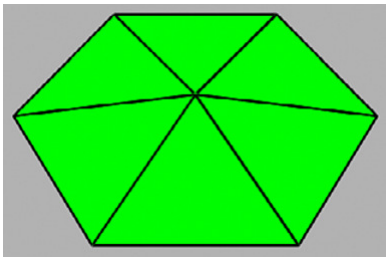
  `glDrawElements(GL_TRIANGLE_FAN,...)`

- Figure shows a fan of three triangles produced by specifying four vertices



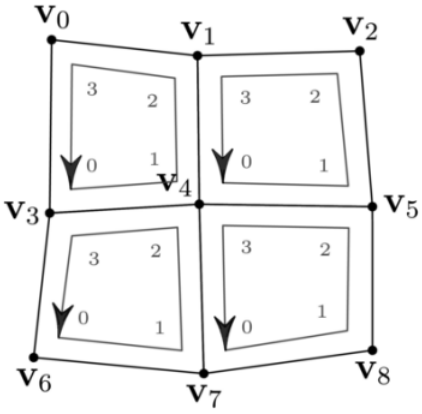
Triangle fans

- The first vertex, V_0, forms the origin of the fan
- After the first three vertices that are used to draw the initial triangle, all subsequent vertices are used with the origin (V_0) and the vertex immediately preceding it (V_{n-1}) to form the next triangle



Quad meshes

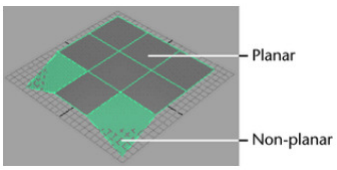
- Similar options as for storing triangles
  - Flat quads, indexed quad meshes, quad strips, no fans



Triangle vs. quad mesh

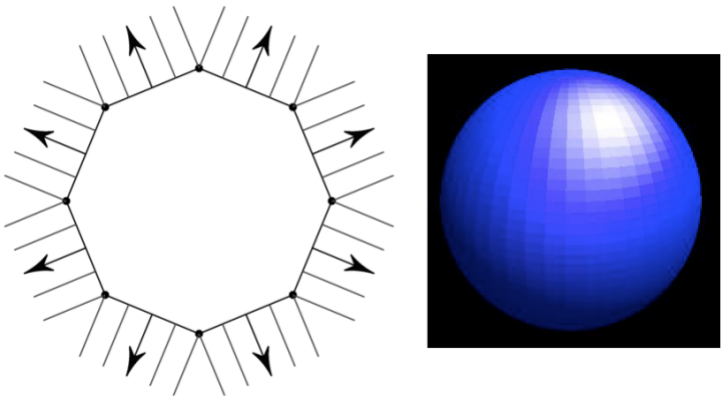
- Triangle mesh
  - Well defined: a triangle is **always planar** and **convex**
  - Irregular arrangement: hard to manipulate for artists
  - Used in rendering as low-level representation
- Quad mesh
  - Not well defined: a quad can be **non-planar** and **concave**
  - Regular arrangement: convenient for modeling
  - Converted to triangles for rendering

quad mesh with planar and non-planar facets



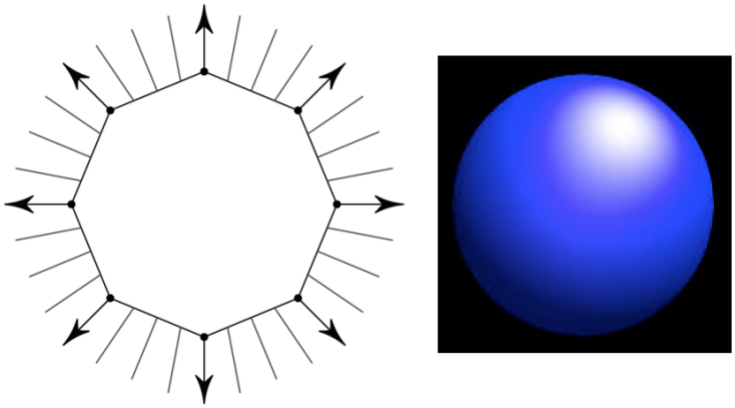
Defining normals

- **Face normal**: same normal for all points in face
  - Geometrically correct, but faceted look



Defining normals

- **Vertex normal**: store normal at vertices, interpolate in face
  - Geometrically "inconsistent", but smooth look



Defining normals

- A **vertex normal** is the average of the normals of adjacent faces
  $\mathbf{n}_i = normalize\left(\sum_{f \in adj(i)} \mathbf{n}_f\right)$
- Computed as

```
foreach vertex: normal[vertex]=(0,0,0)
foreach face
   fnormal = compute_face_normal(face)
   for vertex in face
      normal[vertex] += fnormal
foreach vertex: normal[vertex]=normalize(normal[vertex])
```

- **Triangle** (face) **normal**: $\mathbf{n}_i = normalize\left((\mathbf{b}-\mathbf{a})\times(\mathbf{c}-\mathbf{a})\right)$
- **Quad normal**: average of two triangle normals
  - Only approximated

POLYGONAL MESH FILE FORMAT

Common 3D file formats

- There are many 3D file formats
- Some of the most common are
  - OFF
  - VRML
  - PLY
  - OBJ
  - STL
  - …

OFF file format

- **Object File Format** (`.off`) files are used to represent the geometry of a model by specifying the polygons of the model's surface
  - Polygons can have any number of vertices
- OFF files are all ASCII files
  - begin with the keyword OFF
  - The next line states the number of vertices, the number of faces, and the number of edges
  - The number of edges can be safely ignored
  - The vertices are listed with x, y, z coordinates, written one per line
- After the list of vertices, the faces are listed, with one face per line. For each face, the number of vertices is specified, followed by indices into the list of vertices

OFF file format

```
OFF
numVertices numFaces numEdges
x y z
x y z
... #numVertices like above
NVertices v1 v2 v3 ... vN
MVertices v1 v2 v3 ... vM
... #numFaces like above
```

vertex coordinates

face topology

- Note that vertices are numbered starting at 0 (not starting at 1), and that `numEdges` will always be zero

OFF file example

- A simple example for a cube
- OFF does not support color or texture specifications

```
OFF # a cube
8 6 0
-0.5 -0.5 0.5
0.5 -0.5 0.5
-0.5 0.5 0.5
0.5 0.5 0.5
-0.5 0.5 -0.5
0.5 0.5 -0.5
-0.5 -0.5 -0.5
0.5 -0.5 -0.5
4 0 1 3 2
4 2 3 5 4
4 4 5 7 6
4 6 7 1 0
4 1 7 5 3
4 6 0 2 4
```

VRML file format

- **VRML** (**Virtual Reality Modeling Language**, originally—before 1995—known as the Virtual Reality Markup Language) is a standard file format for representing 3D interactive vector graphics, designed particularly with the WWW in mind
- It has been superseded by **X3D**
- The Web3D Consortium has been formed to further support the collective development of the format
- VRML (and its successor, X3D), have been accepted as international standards by the **International Organization for Standardization** (ISO)

VRML file format

- VRML is a text file format (`.wrl`) where, e.g., vertices and edges for a 3D polygon can be specified along with the surface color, UV mapped textures, shininess, transparency, and so on
- URLs can be associated with graphical components so that a web browser might fetch a webpage or a new VRML file from the Internet when the user clicks on the specific graphical component
- Animations, sounds, lighting, and other aspects of the virtual world can interact with the user or may be triggered by external events such as timers
- A special Script Node allows the addition of program code (e.g., written in Java or ECMAScript) to a VRML file

PLY file format

- **PLY** is a computer file format (`.ply`) known as the **Polygon File Format** or the **Stanford Triangle Format**
- It was principally designed to store 3D data from 3D scanners
- The data storage format supports a relatively simple description of a single object as a list of nominally flat polygons
- A variety of properties can be stored, including: color and transparency, surface normals, texture coordinates and data confidence values
  - The format permits one to have different properties for the front and back of a polygon
- There are two versions of the file format: ASCII and binary

OBJ file format

- **OBJ** is a geometry definition file format (`.obj`) first developed by ***Wavefront Technologies*** for its **Advanced Visualizer** animation package
- The file format is open and has been adopted by other 3D graphics application vendors
- For the most part it is a universally accepted format

OBJ file format

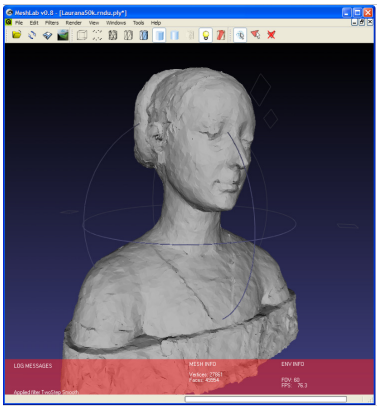
- The OBJ file format is a simple data-format that represents 3D geometry alone — namely, the position of each vertex, the UV position of each texture coordinate vertex, vertex normals, and the faces that make each polygon defined as a list of vertices, and texture vertices
- Vertices are stored in a counter-clockwise order by default, making explicit declaration of face normals unnecessary
- OBJ coordinates have no units, but OBJ files can contain scale information in a human readable comment line

STL file format

- **STL** (**STereoLithography**) is a file format (`.stl`) native to the **stereolithography CAD** software created by **3D System**
  - STL is known also as "Standard Triangle Language" and “Standard Tessellation Language”
- This file format is supported by many other software packages; it is widely used for rapid prototyping, **3D printing** and computer-aided manufacturing
- STL files describe only the surface geometry of a 3D object without any representation of color, texture or other common CAD model attributes
- The STL format specifies both ASCII and binary representations
  - Binary files are more common, since they are more compact

MeshLab

- **MeshLab** is a free and open-source cross-platform application for visualizing, processing, and converting from different three-dimensional meshes file format
- MeshLab is developed by the ISTI-CNR research center in Pisa http://www.meshlab.net/



IMPLICIT SURFACES

Curves

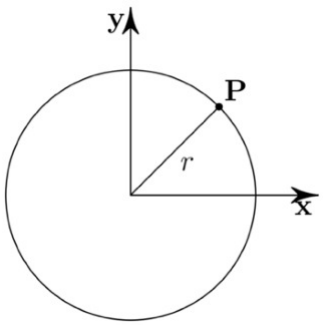
- Used in many contexts
  - Fonts (2D)
  - Animation paths (3D)
  - Shape modeling (3D)
- Different representations
  - **Implicit curves**
  - **Parametric curves**
- 2D and 3D curves are mostly the same

Implicit curve representation

- Implicit **curve** representation

$$f(\mathbf{P})=0$$

e.g., XY circle $f(\mathbf{P})=0 \quad => \quad x^2+y^2-r^2=0$

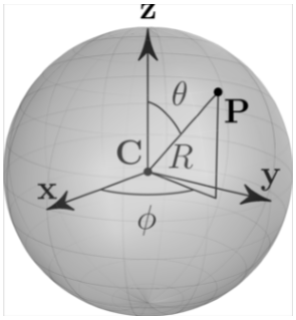


Implicit surface representation

- Implicit **surface** representation

$$f(\mathbf{P})=0$$

e.g., sphere $f(\mathbf{P})=0 \Rightarrow x^2+y^2+z^2-r^2 = 0$



Implicit surfaces

- One way to represent a **smooth surface** is as the set of points that evaluate to zero under some given **trivariate function** $f(x, y, z)$, *i.e.*, $f(x, y, z) = 0$
  This is called an ***implicit representation***

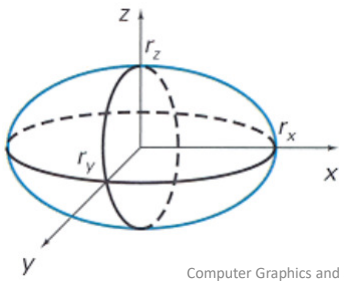
- How do we define implicit functions?
  - **Algebraic**
  - "**Blobby**" models
  - Skeletons
  - Samples
  - Variational

Implicit algebraic surfaces

- For example, simple shapes, like spheres and ellipsoids, can be represented as the **zero set** of a **quadratic trivariate function**
- Implicit function is polynomial

$$f(x, y, z) = ax^d + by^d + cz^d + ex^{d-1}y + fx^{d-1}z + gy^{d-1}x + \ldots$$

$$\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 - 1 = 0$$



Implicit algebraic surfaces

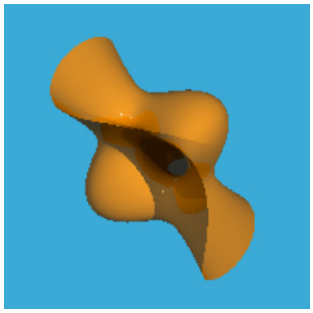
- Most common form: **quadrics**

$$f(x, y, z) = ax^2 + by^2 + cz^2 + 2dxy + 2eyz + 2fxz + 2gx + 2hy + 2jz + k$$

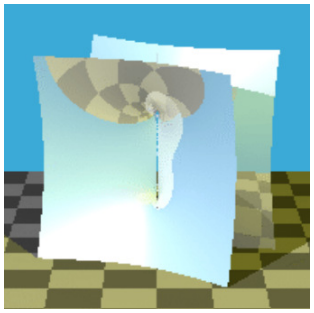
  - Sphere
  - Ellipsoid
  - Paraboloid
  - Hyperboloid

Implicit algebraic surfaces

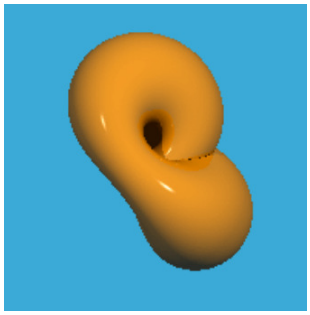
- Higher degree algebraic



Cubic



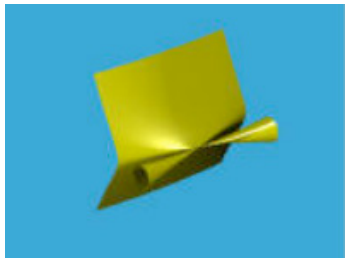
Quartic



Degree six

Implicit algebraic surfaces

- Function extends to infinity
  - Must trim to get desired patch

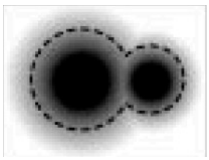


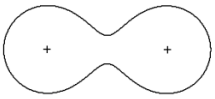
Implicit surfaces: blobby models

- Additionally, we can define an **implicit function** as a sum of simpler **elemental implicit functions**

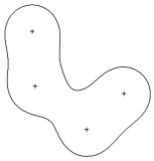
$$f(x, y, z) = \sum_i f_i(x, y, z)$$

- Doing so, say, with a set of **sphere functions** (sum of spherical basis functions), creates a smoothly blended union of the individual spheres
- This technique is known as ***blobby modeling***, and is ideal for modeling organic shapes





Sum of two blobs



Sum of four blobs

Implicit surfaces: blobby models

Implicit function representations are very good at representing **blobby shapes** such as this digital water fountain

Blobby hand
(RenderMan, Pixar)

Blobby models

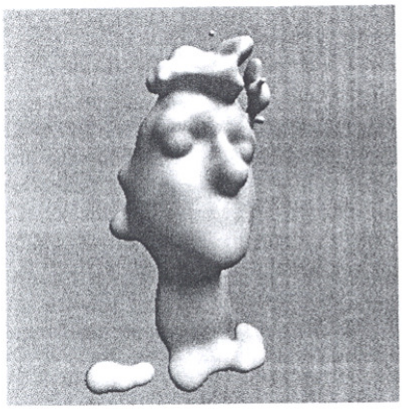
- Blobby molecules $D(r) = ae^{br^2}$
- Meta balls

$$D(r) = \begin{cases} a(1 - \frac{3r^2}{b^2}) & 0 \le r \le b/3 \\ \frac{3a}{2}(1 - \frac{r}{b})^2 & b/3 \le r \le b \\ 0 & b \le r \end{cases}$$

- Soft objects

$$D(r) = \begin{cases} a\left(1 - \frac{4r^6}{9b^6} + \frac{17r^4}{9b^4} - \frac{22r^6}{9b^2}\right) & r \le b \\ 0 & r \ge b \end{cases}$$

Blobby model of face

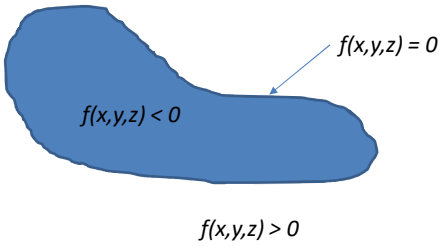


(e) $N = 70$

(f) $N = 243$

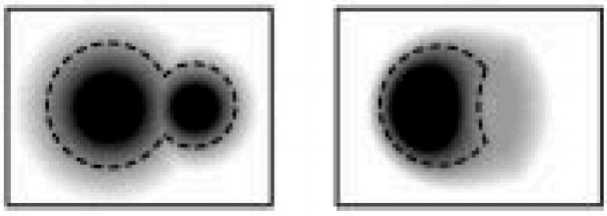
Implicit surfaces properties

- Implicit surfaces have some nice properties
- If we define a surface as the **zero set** of *f*, then we can think of the points where *f* evaluates to a **negative value** as the **volumetric interior** of the surface
  - *f(x,y,z)* < 0 (inside)
  - *f(x,y,z)* = 0 (on surface)
  - *f(x,y,z)* > 0 (outside)



Implicit surfaces properties

- As such, it is easy to apply **volumetric set operations**, such as *union*, *intersection*, *negation* and *difference* to these interior volumes

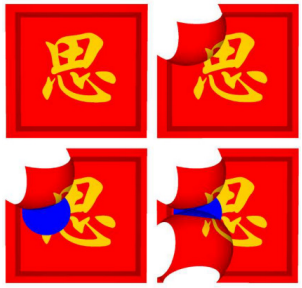


Union Difference

Implicit surfaces properties

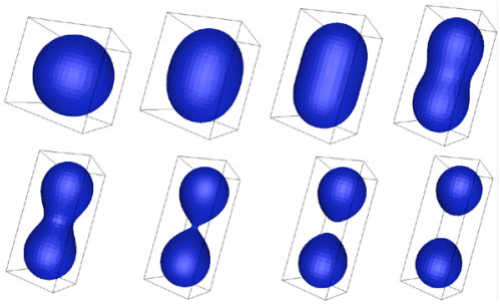
- For example, the **intersection** of the **volumes** defined by functions f_1 and f_2 can be defined using the new function $f(x, y, z) = min(f_1(x, y, z), f_2(x, y, z))$

Volume **intersection** operations have been used to cut out the shown shape



Implicit surfaces properties

- Efficient **topology** changes
  - Surface not represented explicitly



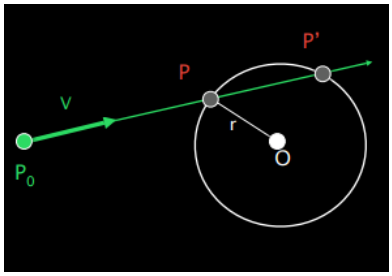
Implicit surfaces rendering

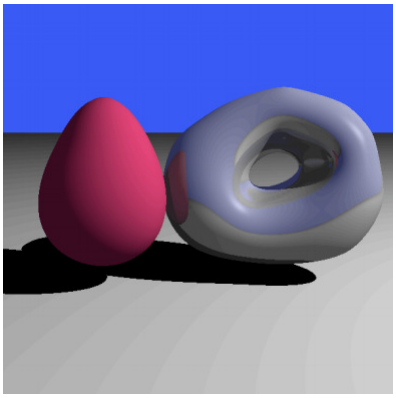
- In order to **render an implicit surface** in OpenGL, we need to create a set of triangles that approximate the implicit surface
- This is a non-trivial task
  - Polygonization
- Alternatives
  - Ray tracing
  - Contours
  - Floating particles

Implicit surfaces rendering

- A possibility is using **ray tracing** for rendering (efficient computation of ray-surface intersection)

Ray: $P = P_0 + tV$
Sphere: $|P - O|^2 - r^2 = 0$
Substituting for P, we get: $|P_0 + tV - O|^2 - r^2 = 0$
Solve quadratic equation: $at^2 + bt + c = 0$





Implicit surface summary

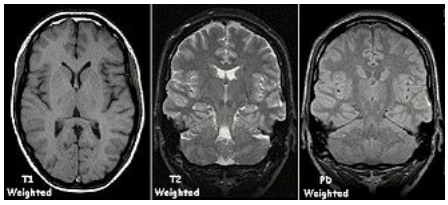
- Advantages
  - Easy to test if point is on surface
  - Easy to compute intersections / unions / differences
  - Easy to handle topological changes
- Disadvantages
  - Indirect specification of surface
  - Hard to describe sharp features
  - Hard to enumerate points on surface => slow rendering

(Implicit) Volume

- A **volume representation** is a specific kind of implicit representation that uses a regular 3D grid of discrete values called ***voxels***
- Using **trilinear interpolation over each grid cell**, these voxels define a **continuous function** in 3D
- The **zero set** of this function can then be thought of as an **implicit surface**

Volume

- Volume data are often obtained as the output of a volumetric scanning process, such as a **Magnetic Resonance Image** (MRI)

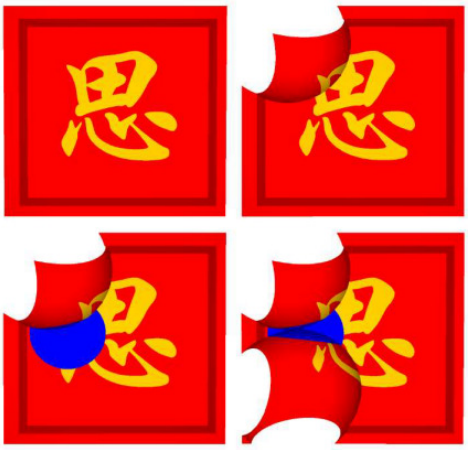


- We can also take some general implicit function representation and then **sample** it along a **regular 3D grid** to get a voxel-based approximation

Volume

- To **render** the **zero-set of a volume** based representation in OpenGL, one needs to extract a set of **triangles that approximate the zero-set**
- Due to the regular pattern of the data, this can be done a bit more easily than for a general implicit function
- The standard technique is called the ***marching cube*** method [8], but there are newer methods such as ***dual contouring*** [9] that may give better results (see next Figure)

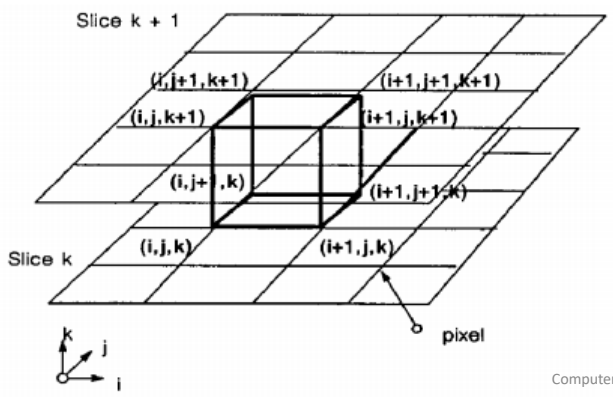
Volume



This surface has been extracted from a volumetric data representation using the **dual-contouring** method.
Volume intersection operations have been used to cut out the shown shape

Marching cubes

- Marching cubes use a divide-and-conquer approach to **locate the surface** in a **logical cube** created from **eight pixels**
  - four each from two adjacent **slices**

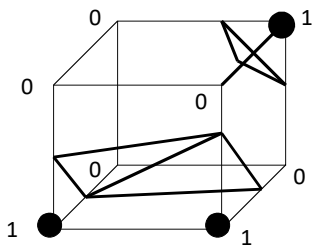


Marching cubes

- The algorithm proceeds through the scalar field, taking eight neighbor locations at a time (thus forming an imaginary cube), then determining the **polygon**(s) needed to **represent** the part of the **isosurface** that passes through this cube
- The individual polygons are then fused into the desired surface
- The algorithm determines how the **surface intersects this cube**, then moves (or marchs) to the next cube

Marching cubes

- To find the surface intersection in a cube
  - Assign a 0 to a cube's vertex if the data **value at that vertex exceeds** (or equals) the **value of the surface** we are constructing. These vertices are **inside** (or on) the surface
  - Cube vertices with **values below the surface** receive a **1** and are **outside the surface**
  - The **surface intersects those cube edges** where **one vertex is outside the surface** (1) and the other is **inside the surface** (0). With this assumption, we determine the topology of the surface within a cube, finding the location of the intersection later

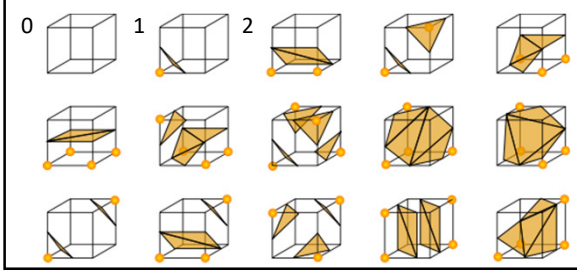


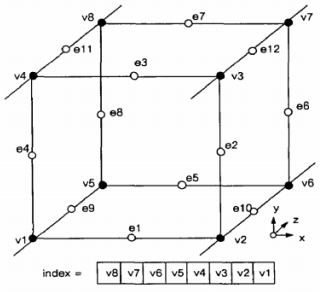
Marching cubes

- Since there are eight vertices in each cube and two states, **inside** and **outside**, there are only $2^8 = 256$ ways a surface can intersect the cube (treating each of the 8 scalar values as a bit in an 8-bit integer)
- By enumerating these 256 cases, a **table** is created to **look up surface-edge intersections**, given the **labeling of a cubes vertices**
- The table contains the edges intersected for each case
- Triangulating the 256 cases is possible, but tedious and error-prone
  - Two different symmetries (**symmetry** 256/2=128, **rotation** 128/8=16) of the cube reduce the problem from 256 cases to 15 patterns

Marching cubes

- The simplest pattern, 0, occurs if **all vertex values are above** (or **below**) the selected value and produces no triangles
- The next pattern, 1, occurs if the **surface separates on vertex from the other seven**, resulting in **one triangle defined by the three edge intersections**
- Other patterns produce multiple triangles
- Permutation of these 15 basic patterns using complementary and rotational symmetry produces the 256 cases





Marching cubes

- The final value, after all eight scalars are checked, is the **actual index to the polygon indices array**
- Finally, **each vertex** of the generated polygons is placed on the appropriate position along the cube's edge by **linearly interpolating** the two **scalar values that are connected by that edge**
- The gradient of the scalar field at each grid point is also the normal vector of a hypothetical isosurface passing from that point
- Therefore, these **normals may be interpolated** along the edges of each cube to find the **normals of the generated vertices**, which are essential for shading the resulting mesh with some illumination model

Marching cubes

- In summary, marching cubes create a surface from a 3D set of data as follows
  1. Read four slices into memory
  2. Scan two slices and create a cube from four neighbors on one slice and four neighbors on the next slice
  3. Calculate an index for the cube by comparing the eight density values at the cube vertices with the surface constant
  4. Using the index, look up the list of edges from a pre-calculated table
  5. Using the densities at each edge vertex, find the surface edge intersection via linear interpolation
  6. Calculate a unit normal at each cube vertex using central differences. Interpolate the normal to each triangle vertex
  7. Output the triangle vertices and vertex normal

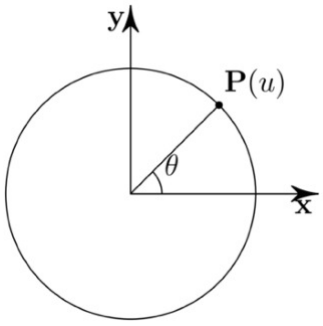
PARAMETRIC PATCHES

Parametric curve representation

- Parametric **curve** representation

$$\mathbf{P}(u) = (f_x(u), f_y(u), f_z(u))$$

e.g., XY circle $\mathbf{P}(u) = (r\cos\theta, r\sin\theta, 0)$



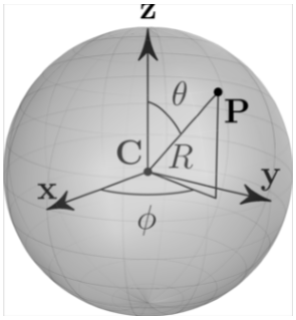
Parametric surface representation

- Parametric **surface** representation

  $\mathbf{P}(u) = (f_x(u), f_y(u), f_z(u))$

  e.g., sphere $\mathbf{P}(u) = (r\cos\phi\sin\theta,\ r\sin\phi\sin\theta,\ r\cos\theta)$

  where u=(r, ϕ, θ)



Parametric surface representation

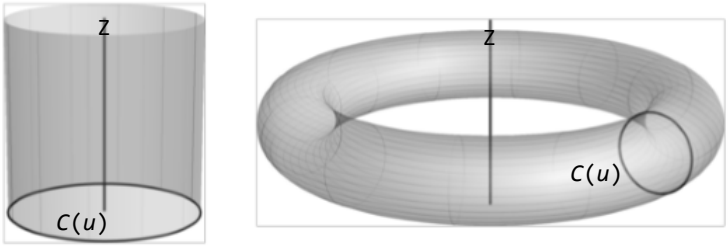
- Goals when defining
  - Smoothness, efficiency, local control
  - Same as curves
- Two solutions

  (1) **Combine curves** to **obtain surfaces**
  - Easy for simple solids, but not general

  (2) **Extend parametric curves** to **surface patches**
  - General formulation
  - Used heavily in CAD

Surfaces from curves (1)

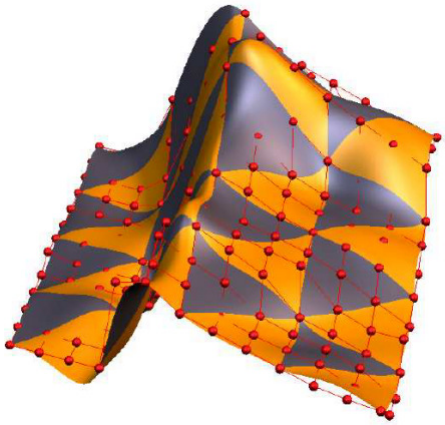
- Example: **Extrude** curve $C(u)$ in the XY-plane along Z-axis
  $\mathbf{P}(u,v) = (C_x(u),\ C_y(u),\ v)$
- Example: **Revolve** curve $C(u)$ in YZ-plane about Z-axis
  $\mathbf{P}(u,v) = (C_x(u)\cos v,\ C_y(u)\sin v,\ C_z(u))$



Parametric patches (2)

- **Parametric patches** represent a **section of surface** called a ***patch*** using 3 coordinate functions *x(u, v)*, *y(u, v)* and *z(u, v)*
- These functions are **defined over some square** or **triangular portion** of the (*u, v*) **plane**
- In most cases, each coordinate function over the patch is represented as a **piecewise polynomial bivariate function** (see next slide)

Parametric patches



A spline patch controlled by a *m*-by-*n* rectilinear control mesh [10]

Parametric patches

- The most common parametric representation is the ***tensor-product spline surface***
- Recall (lecture 9-C) that a **spline curve** can be used to represent a **curve in space**
- Such a spline curve is defined by an **input control polygon** that **connects a sequence of discrete points in space**
- This **spline curve** is made up of a set of **smaller pieces**, each having coordinate functions that are, say, **cubic polynomials**

Parametric patches

- **Spline curves**: 1D blending functions

$$\mathbf{P}(t) = \sum_i b_i(t)\mathbf{P}_i$$

- **Surface patches**: 2D blending functions cross product of 1D blending functions

$$\mathbf{P}(u,v) = \sum_{ij} b_{ij}(u,v)\mathbf{P}_{ij}$$

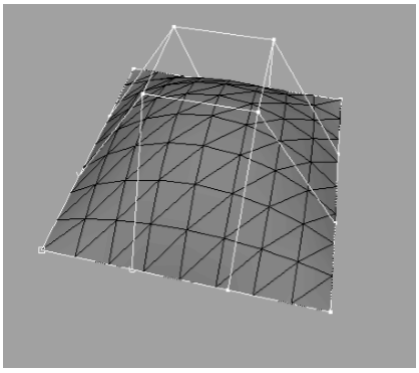
$$b_{ij}(u,v) = b_i(u)b_j(v)$$

Parametric patches

- For a **tensor product spline**, this construction is "upgraded " from **curves to surfaces**
- In this case, the control polygon is replaced by an *m*-by-*n* rectilinear ***control mesh***, with vertices in 3D
- By appropriately applying the spline definitions in both the *u* and *v* variables, we end up with a parametric patch
- Such a patch is made up of small pieces defined over small squares in the (*u, v*) domain
- The coordinates of each such piece are polynomials in *u* and *v*

Parametric patches

- In the case where we upgrade from a **cubic spline curve construction**, each **square piece is defined by a bi-cubic polynomial** in (*u, v*) (i.e., have terms up to the highest power $u^3 v^3$)



Bicubic Bezier patch

Parametric patches

- Just like curves
  - Uniform
  - Non-uniform
- Non-uniform rational B-splines (NURBS)
  - Ratios of B-splines
  - Invariance under perspective
  - Can represent conic sections exactly
  - Often used in 3D

Parametric patches rendering

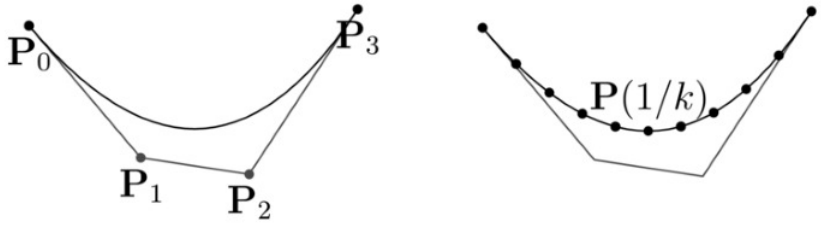
- For rendering, splines can easily be approximated by a **quad mesh** in a number of ways
- Meshes are efficient to draw in hardware and software
  - More faces to provide better approximation
- Two common solutions include **uniform** and **adaptive tessellation**

Uniform tessellation

- For example, we can just place a **fine grid of sample points** over the (*u, v*) domain and **evaluate the spline functions** to obtain $[x, y, z]^t$ coordinates for each sample
  - Fast to compute and simple to implement
  - Generates many segments
- These **samples** can be used as the **vertices of a regular quad mesh**

Uniform tessellation

- Split into k segments uniformly at $t_k=1/k$



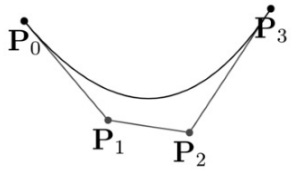
Bezier curve with 4 control points

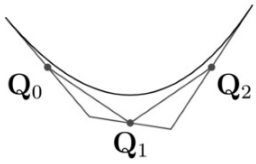
Adaptive tessellation

- Another approach is to apply a recursive refinement process that takes in a **control mesh** and outputs a **denser control mesh** that represents the same surface
- After a **few steps of subdivision**, the quads themselves form a dense control mesh that forms a **good approximation** to the **underlying spline surface**
- Bezier patches can use ***De Castaljeu's*** algorithm

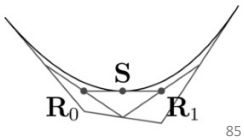
Adaptive tessellation

- **De Casteljau** algorithm: **recursively split to small splines**
  - If flat enough: draw control segments
  - Otherwise, split each control segment $\mathbf{Q}_i = (\mathbf{P}_i + \mathbf{P}_{i+1})/2$





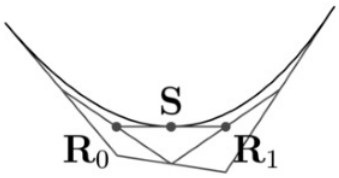
  - Split new segment $\mathbf{R}_i = (\mathbf{Q}_i + \mathbf{Q}_{i+1})/2$
  - Split again $\mathbf{S} = (\mathbf{R}_i + \mathbf{R}_{i+1})/2$

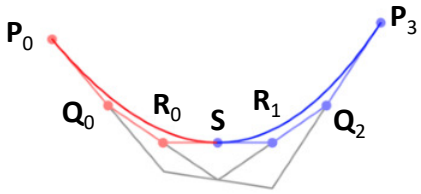


85

Adaptive tessellation

- Two **Bezier splines** {$\mathbf{P}_0$. $\mathbf{Q}_0$, $\mathbf{R}_0$, **S**} (in red below), {**S**, $\mathbf{R}_1$, $\mathbf{Q}_2$, $\mathbf{P}_3$} (in blue below)
- Recursive algorithm as above





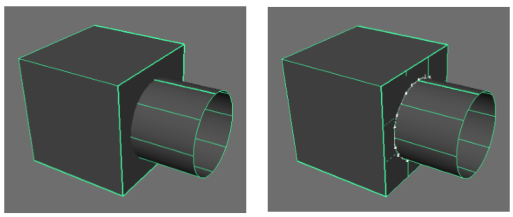
Computer Graphics and 3D 86

Parametric patches

- One can typically design surfaces using splines in a geometric modeling package
- Spline surfaces are very popular in the Computer Aided Design (CAD) community
- In particular, due to their explicit parametric polynomial representation, it is easy to apply various calculations to such shapes

Parametric patches

- Modeling with splines is not without a number of difficulties
- To **model a closed** (say ball-like) **surface**, we have to **stitch together some number of patches**
- Doing the stitching in a smooth manner is tricky
- Additionally, if we want patches to meet up along specific creased curves, we need to explicitly do something called "***trimming***" to each of the patches



- One interesting book about the spline representation is [11]

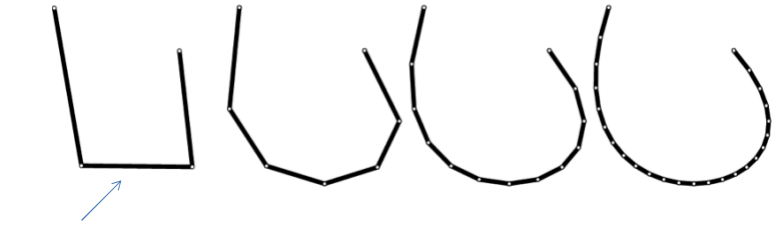
Parametric vs. implicit

- **Implicit**
  - Efficient intersections and topology changes
- **Parametric**
  - Efficient "**enumeration**" of surface and rendering

SUBDIVISION SURFACES

Subdivision curves

- Start with a **piecewise linear curve**
- Apply recursively **subdivision rule**



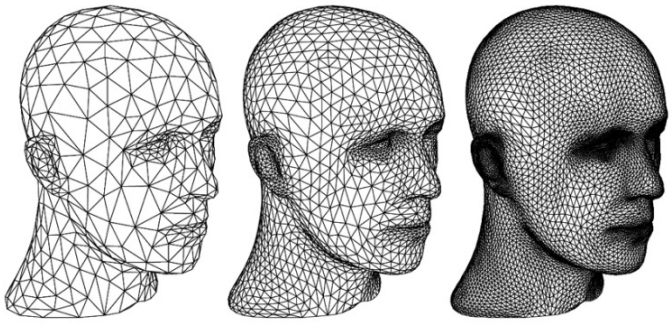
Piecewise linear

Apply recursive subdivision

Increasing smoothness

Subdivision surfaces

- Start with a **polygon mesh**
- Apply recursively **subdivision rule**



Polygon mesh

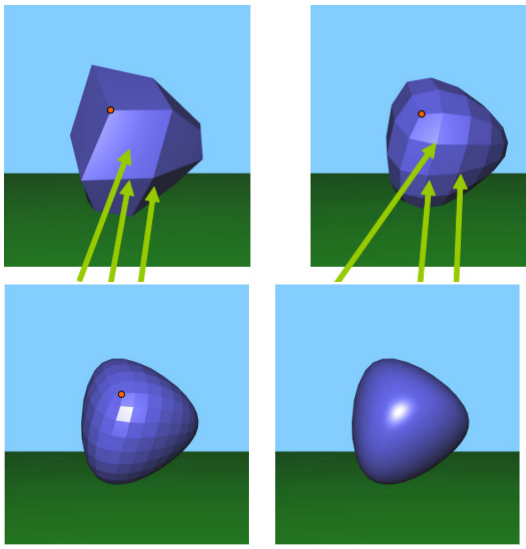
Apply recursive subdivision

Increasing smoothness

Subdivision surfaces

- The ***subdivision surface*** is a simple representation that solves many of the difficulties associated with parametric patches
- The basic idea is to start with a single **control mesh**
- This mesh needs not be rectilinear; it can represent a closed surface and can have vertices of any valence
- We then use a set of rules to **refine the mesh**, resulting in a **higher resolution control mesh**
- By applying this process **recursively** some number of times, we obtain a **very high resolution mesh** that can be directly rendered as an **approximation to a smooth surface**

Subdivision surfaces



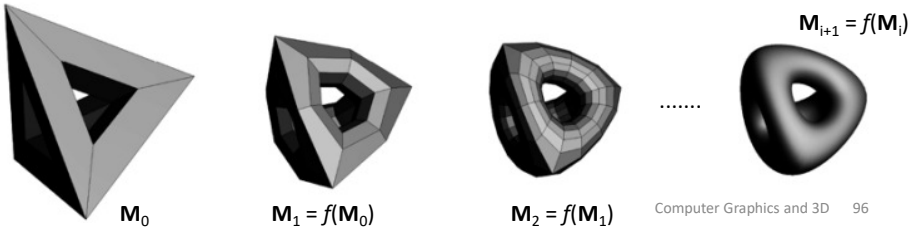
On the left, we show one more level of subdivision.
On the right, the same mesh has been smooth shaded

Subdivision surfaces

- Subdivision surfaces **do not require any patching steps**
- Additionally, **special rules** can be applied along certain portions of the control mesh to achieve **creases in the surface** where desired
- The subdivision rules are defined such that the meshes arising after more and more levels of subdivision converge to a smooth ***limit-surface*** (a C^1 immersed submanifold in R^3)
- In many cases, such as in regions where the mesh is rectilinear, the subdivision steps match those used to **refine a tensor-product spline surface**

Subdivision surfaces

- A subdivision surface is the **limit surface** of a subdivision process
  - Provably smooth
- Subdivision process
  - Start with input mesh $\mathbf{M}_0$
  - Recursively apply subdivision rule f: $\mathbf{M}_{i+1} = f(\mathbf{M}_i)$
- Limit surface $\mathbf{S} = \lim_{i\to\infty} \mathbf{M}_i = \lim_{i\to\infty} f^i(\mathbf{M}_0)$



$\mathbf{M}_0$ $\mathbf{M}_1 = f(\mathbf{M}_0)$ $\mathbf{M}_2 = f(\mathbf{M}_1)$

Subdivision surfaces

- One drawback of a subdivision surface is that the **surface representation is defined by a procedure**, and not by a **formula**
- Thus, such surfaces are **more difficult to mathematically analyze**
- Additionally, away from the rectilinear regions (at so-called ***extraordinary points***) the limit surface can be somewhat poorly behaved [12] and harder to control
- But, for casual use (movies and games), these are not big issues, and subdivision surfaces have proven to be **simple**, **useful**, and **general**

Implementing subdivision surfaces

- Implementing subdivision surfaces is notoriously hard in standard formulation
  - Complex rules, requires data structures for adjacency
- Reformulate as **successive averaging** based on a factored approach [13]
  - Subdivision requires more averaging passes to adjust position
  - Uses only an edge hash table: non complex data structure
  - **Loop**, **Catmull-Clark**, mixed triangle/quad, creases

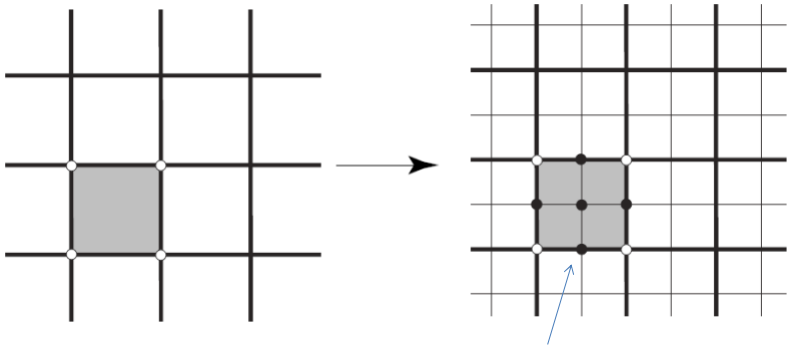
Subdivision surfaces

- Two stages
  1. Refine mesh (mesh subdivision)
  2. Place vertices
- **Mesh subdivision: new topology**
  - Create new vertices and faces
- **Vertex placement: new geometry**
  - Compute vertex position

- Different schemes available
  - Depend on input/output topology and geometry
- ***Catmull-Clark***: quad meshes
- ***Loop***: triangle meshes

Catmull-Clark: step 1 (topology change)

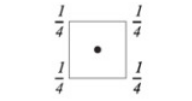
- **Mesh subdivision**: each quad is split in four quads



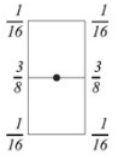
New vertices and faces does appear
(**topology change**)

Catmull-Clark: step 2 (geometry change)

- **Vertex placement**: depends on where vertex originated

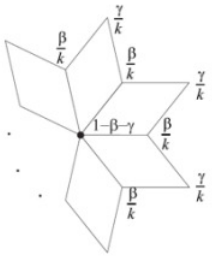


Mask for a face vertex



Mask for an edge vertex

Interior

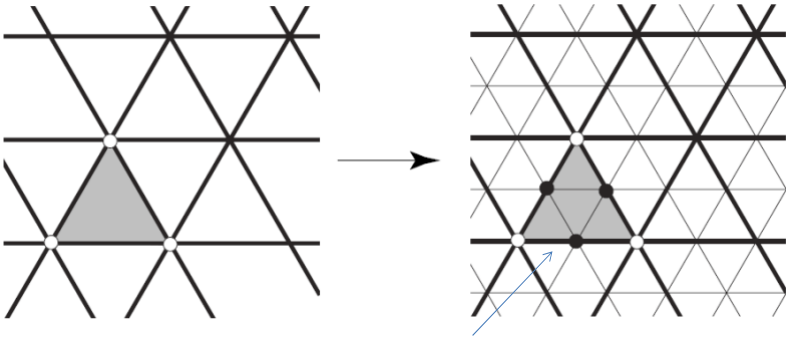


Mask for a boundary odd vertex

Crease and boundary

Loop: step 1 (topology change)

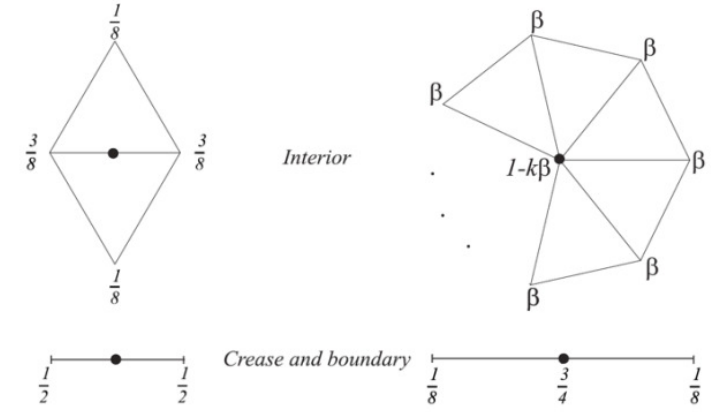
- **Mesh subdivision**: each triangle is split in four triangles



New vertices and faces does appear
(**topology change**)

Loop: step 2 (geometry change)

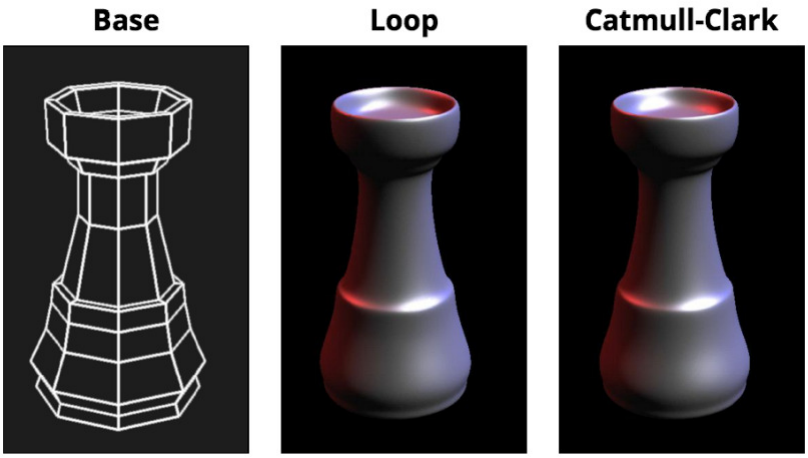
- **Vertex placement**: depends on where vertices originated



a. *Masks for odd vertices*

b. *Masks for even vertices*

Loop vs. Catmull-Clark

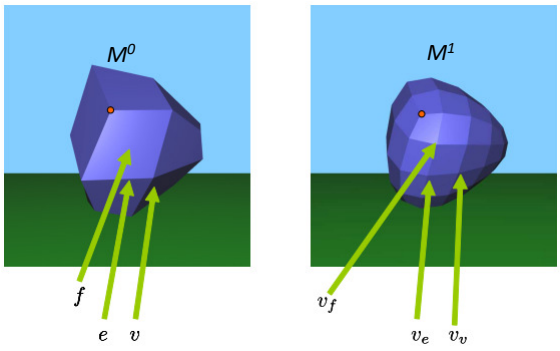


Catmull-Clark

- Here we describe in more detail a specific subdivision representation due to ***Catmull*** and ***Clark***
- One starts with some input mesh M^0 ,which we will assume is a **watertight** "**manifold with no boundary**"
- This mesh has **connectivity** (**topological**) **information** describing its vertex, edge and face structure
- The mesh also has **geometric information**, mapping each abstract vertex to a point in 3D

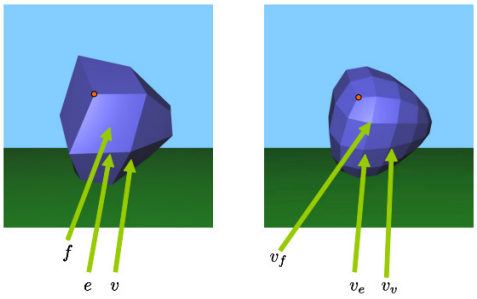
Catmull-Clark

- Now we apply a set of connectivity updates to get a new refined mesh M^1, with its own connectivity and geometry
- The connectivity of M^1 is defined as follows
  - For each vertex v in M^0, we associate a new "**vertex-vertex**" v_v in M^1
  - For each edge e in M^0 we associate a new "**edge-vertex**" v_e in M^1
  - For each face f in M^0 we associate a new "**face-vertex**" v_f in M^1



Catmull-Clark

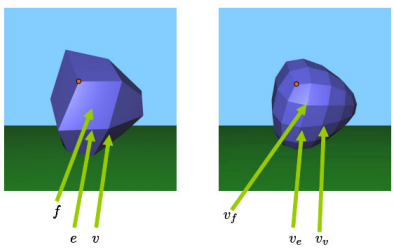
- These new vertices are connected up with new edges and faces



- We can easily verify that in M^1, **all faces are quads**
- In M^1 we call any vertex of valence four "***ordinary***" and any vertex of valence different from four "***extraordinary***"

Catmull-Clark

- We apply this subdivision process **recursively**
- Given M^1, or more generally, M^i, for any $i \geq 1$, we apply the same subdivision rules to obtain a finer mesh M^{i+1}
- In the new mesh, we have roughly 4 times the number of vertices
- But, importantly, we can verify that the **number of extraordinary vertices stays fixed**
- Thus, during subdivision, **more and more of the mesh looks locally rectilinear**, with a **fixed number of isolated extraordinary points** in between



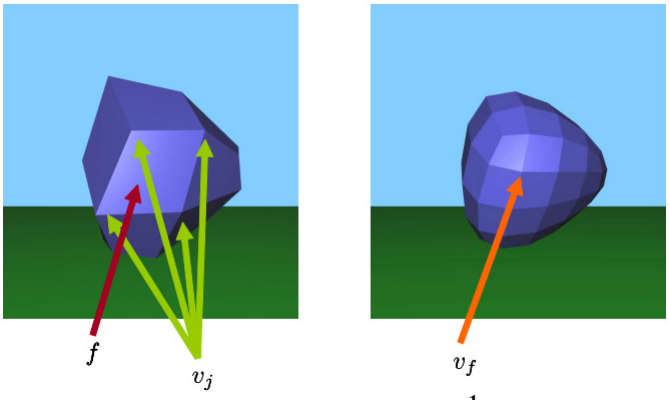
Catmull-Clark

- Now all we need are rules to determine the **geometry** for the **new vertices** as they are created during each subdivision step
- First, let *f* be a face in M^i surrounded by the vertices v_j (and let m_f be the number of such vertices)
- We set the geometry of each new ***face-vertex*** v_f in M^{i+1} to be

$$v_f = \frac{1}{m_f}\sum_j v_j \qquad (14.1)$$

i.e., the centroid of the vertices in M^i defining that face (again, for any subdivision level $i \geq 1$, we have $m_f = 4$)

Catmull-Clark



$$v_f = \frac{1}{m_f}\sum_j v_j$$

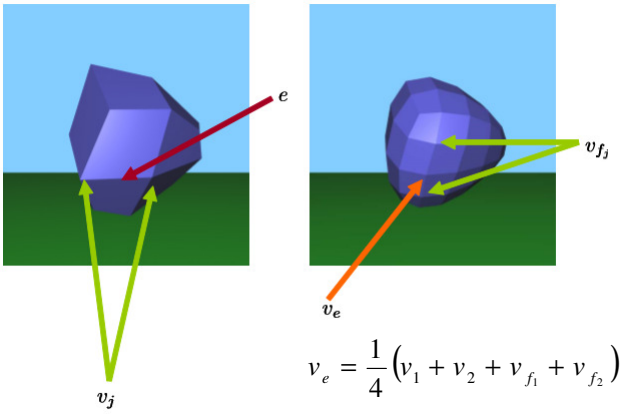
A face *f* on the left gives rise to v_f on the right.
The geometry of v_f is determined by the v_j on the left

Catmull-Clark

- Next, let *e* be an edge in M^i connecting the vertices v_1 and v_2, and separating the faces f_1 and f_2
- We set the geometry of the new ***edge-vertex*** in M^{i+1} to be

$$v_e = \frac{1}{4}\left(v_1 + v_2 + v_{f_1} + v_{f_2}\right) \qquad (14.2)$$

Catmull-Clark



$$v_e = \frac{1}{4}\left(v_1 + v_2 + v_{f_1} + v_{f_2}\right)$$

An edge *e* on the left gives rise to v_e on the right. The geometry of v_e is determined by the v_j on the left and v_{fj} on the right

Catmull-Clark

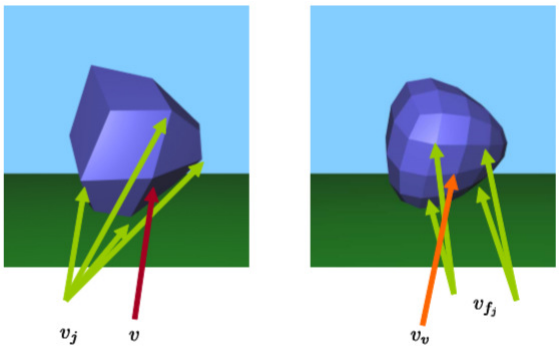
- Finally, let v be a vertex in M_i connected to the n_v vertices v_j and surrounded by the n_v faces f_j
- Then, we set the geometry of the new ***vertex-vertex*** in M_{i+1} to be

$$v_v = \frac{n_v - 2}{n_v} v + \frac{1}{n_v^2} \sum_j v_j + \frac{1}{n_v^2} \sum_j v_{f_j} \qquad (14.3)$$

- For an ordinary vertex, with valence $n_v = 4$, this becomes

$$v_v = \frac{1}{2} v + \frac{1}{16} \sum_j v_j + \frac{1}{16} \sum_j v_{f_j}$$

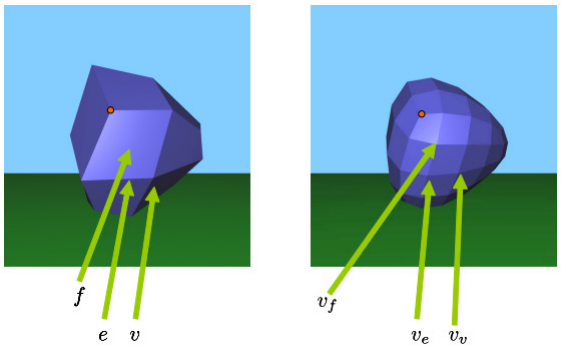
Catmull-Clark



A vertex v on the left gives rise to v_v on the right. The geometry of v_v is determined by the vertices v_j on the left and v_{fj} on the right

$$v_v = \frac{1}{2} v + \frac{1}{16} \sum_j v_j + \frac{1}{16} \sum_j v_{f_j}$$

Catmull-Clark



A vertex *v* on the left gives rise to a vertex v_v on the right.
An edge *e* on the left gives rise to a vertex v_e on the right.
A face *f* on the left gives rise to a vertex v_f on the right.
An **extraordinary vertex** is shown in red (valence 3).
The number of such vertices remain fixed through subdivision

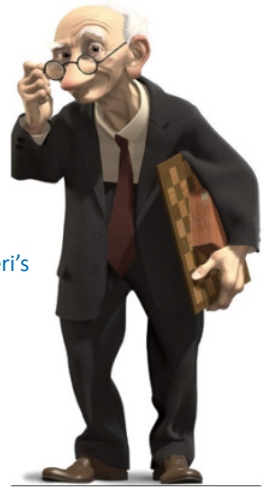
Catmull-Clark

- There has been much study into the development of these **subdivision rules**, and understanding their **convergence properties**
- The first complete **analysis of the first order behavior at extraordinary points** is found in [14]
- In practice, you don't need to know any of this; all you need to do to use subdivision surfaces is to implement Eq.(14.1), (14.2) and (14.3)
- The hardest part is simply getting a mesh data structure set up so that you can implement this computation

Subdivision surface usage

- Heavily used in CG
  - Old algorithms, but practicalities introduced recently
  - Works well with animation
  - Arbitrary topology, no stitching
- Pixar's Geri's Game 1998
  - Catmull-Clark subdivision
  - Complex control meshes
  - Heavy use of creases

Pixar's Geri's game



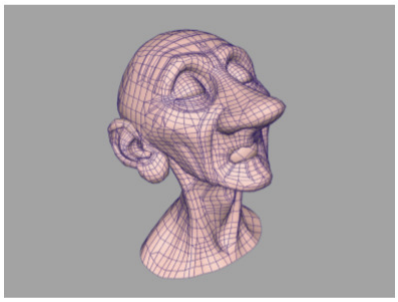
(c) Pixar/Disney

Geri's game

- Creation of believable and endearing characters in CG presents a number of technical challenges, including the **modeling**, **animation** and **rendering** of complex shapes such as heads, hands, and clothing [15][16]

https://www.youtube.com/watch?v=gLQG3sORAJQ

Geri's game



The **control mesh** for Geri's head, created by digitizing a full-scale model sculpted out of clay

Geri's hand as a piecewise smooth **Catmull-Clark surface**. Infinitely sharp creases are used between the skin and the finger nails

Surface representation comparison

	meshes	implicit	parametric	subdiv
accurate	no	yes	yes	yes
concise	no	yes	yes	yes
intuitive specification	no	no	yes	no
local support	yes	no	yes	yes
affine invariant	yes	yes	yes	yes
arbitrary topology	yes	no	no	yes
continuity	no	yes	yes	yes
parameterization	no	no	yes	no
efficient display	yes	no	yes	yes
efficient intersection	no	yes	no	no

References

[1] Steven J. Gortler, “Foundations of 3D Computer Graphics,” The MIT Press, 2012

[2] John F. Hughes et al., “Computer Graphics, Principles and Practice,” Wiley and Sons, Third Edition, 2014

[3] GALLIER, J., “Geometric Methods and Applications: for Computer science and Engineering,” Springer-Verlag, 2001

[4] WARREN, J., and WEIMER, H., “Subdivision methods for geometric design: A constructive approach,” Morgan Kaufmann Pub, 2002

[5] BOTSCH, M., KOBBELT, L., PAULY, M., ALLIEZ, P., and L´EVY, B., “Polygon mesh processing,” AK Peters Ltd., 2010

[6] WRIGHT, R., and LIPCHAK, B., “OpenGL superbible,” 4th ed. Addison-Wesley Professional, 2007

[7] RTWH. Openmesh web page. http://www.openmesh.org

References

[8] LORENSEN, W., and CLINE, H., “Marching cubes: A high resolution 3D surface construction algorithm,” *Proc. of the 14th annual conference on Computer graphics and interactive techniques* (1987), ACM, pp. 163–169

[9] JU, T., LOSASSO, F., SCHAEFER, S., and WARREN, J., “Dual contouring of hermite data,” *ACM Transactions on Graphics* (TOG) 21, 3 (2002), 339–346

[10] REIS, G., ZEILFELDER, F., HERING-BERTRAM, M., FARIN, G., and HAGEN, H., “High-quality rendering of quartic spline surfaces on the GPU,” *IEEE Trans. on Visualization and Computer Graph*ics 14, 5 (2008), 1126–1139

[11] RAMSHAW, L., “Blossoming: A connect-the-dots approach to splines,” *Digital Systems Research Center*, 1987

[12] PETERS, J., and REIF, U., “Shape characterization of subdivision surfaces–basic principles,” *Computer Aided Geometric Design* 21, 6 (2004), 585–599

[13] J. Warren and S. Schaefer, "A factored approach to subdivision surfaces," in *IEEE Computer Graphics and Applications*, vol. 24, no. 3, pp. 74-81, May-June 2004

References

[14] REIF, U., "A unified approach to subdivision algorithms near extraordinary vertices," *Computer Aided Geometric Design* 12, 2 (1995), 153–174

[15] Denis Zorin, Peter Schroder, "Subdivision for Modeling and Animation," SIGGRAPH 2000 Course Notes

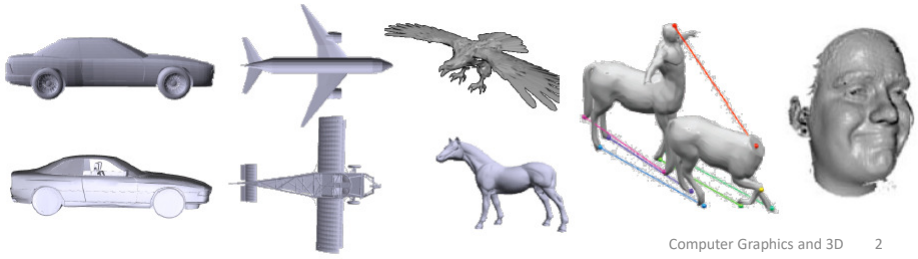
[16] Tony De Rose, Michael Kass, Tien Truong, "Subdivision Surfaces in Character Animation," Conference on Computer Graphics and Interactive Techniques SIGGRAPH '98, pages 85-94

3D Shape Analysis

Lecture 16-A

Problem statement

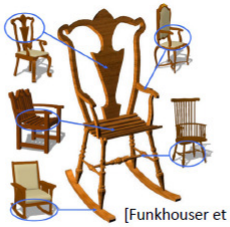
- Shape analysis can serve to solve the shape similarity problem
  - Are two pieces of 3D geometry similar ?
  - How much similar / dissimilar they are ?
  - What are the similar parts ?
- Why is it important ?
- Why in 3D ? Why is it difficult ?



Why is it important ?

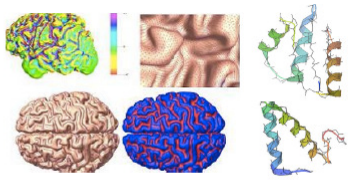
Many applications require estimating similarities between geometric shapes

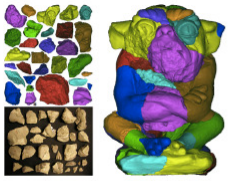
Biometrics (2D, 3D)



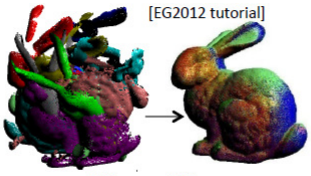
[Funkhouser et al. 04]

3D modeling



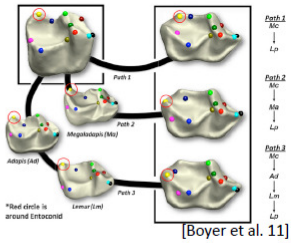


Archaeology



[EG2012 tutorial]

3D acquisition
and scan registration



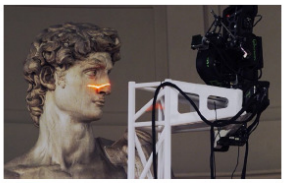
[Boyer et al. 11]

Medicine, biology, chemistry

Why in 3D ?

3D acquisition technologies are widely accessible

[Vlasic et al. 09]



Stanford

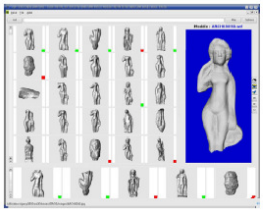
PointGrey Microsoft

3D face reconstruction
[Beeler et al. 10, Shlizerman and Seitz 11]

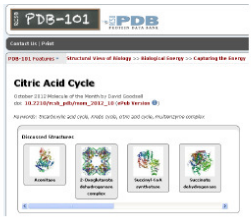
Building Rome in a day

Why in 3D ?

Large repositories of 3D data being created

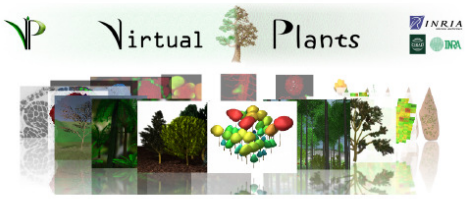


[Gorisse et al. 3DTV2007]



Protein databanks

Trimble 3D Warehouse (Google 3D warehouse)



Virtual plants

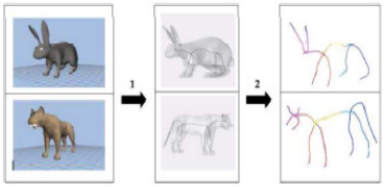
Organize this large amount of 3D data

- Indexation, classification, recognition and retrieval require a **measure of similarity** (distance) for comparing 3D shapes
- What to compare ?

<table>
<tr><td colspan="2">Geometry</td><td colspan="2">Topology</td><td rowspan="7">Whole vs. whole</td><td rowspan="7">Part vs. part / whole</td></tr>
<tr><td colspan="4">Geometry + Topology</td></tr>
<tr><td colspan="4">Semantics</td></tr>
<tr><td colspan="2">Generic</td><td colspan="2">Domain-specific</td></tr>
<tr><td colspan="4">Deformations</td></tr>
<tr><td>Rigid</td><td>Isometric (non-elastic)</td><td colspan="2">Elastic</td></tr>
</table>

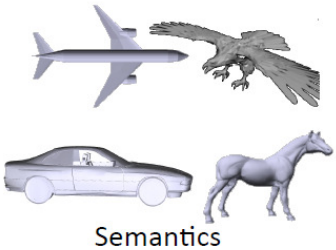
What to compare ?

Geometry



Structure (topology)

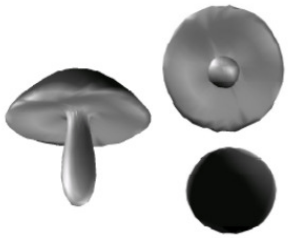
Geometry + Topology



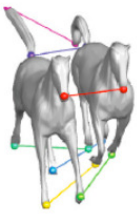
Semantics

Why is it difficult ?

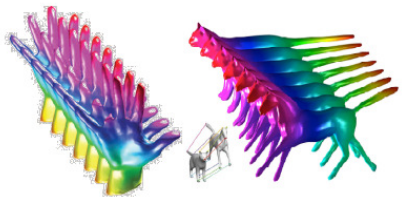
- **Deformations**: a shape is what is left after all deformations are factored out



Rigid transformations
(translation, scale, rotation)



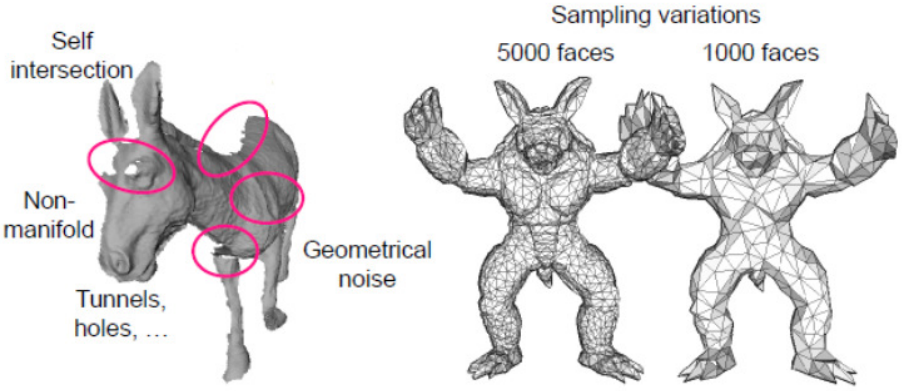
Isometric
deformations



Elastic deformations

Why is it difficult ?

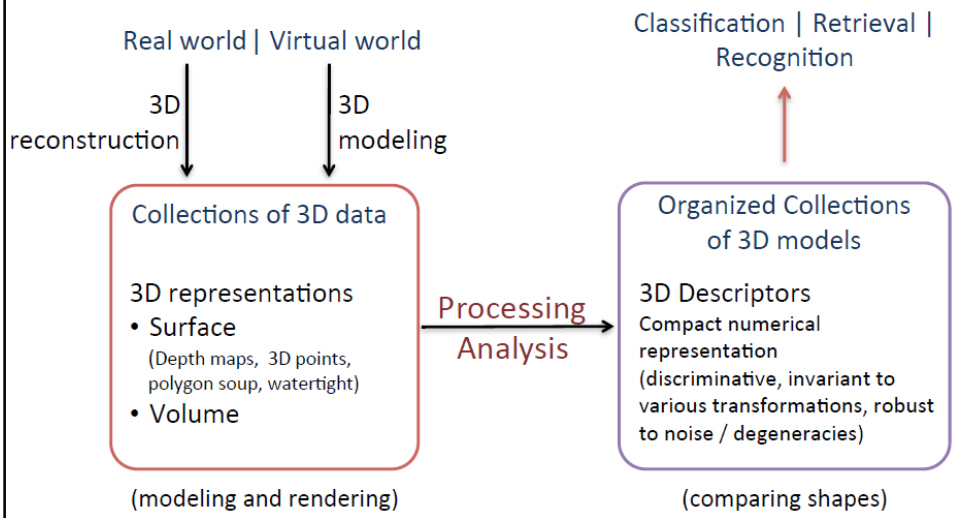
- **Noise**: geometric, topological and sampling noise



Geometric / topological noise

Sampling

Pipeline

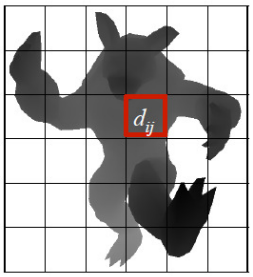


(modeling and rendering)

(comparing shapes)

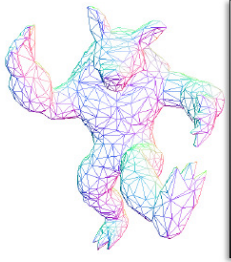
Representations for modeling & rendering

- Surface representations



Depth map
(multiview stereo systems, Kinect, ...)

Point clouds
(range scanners)

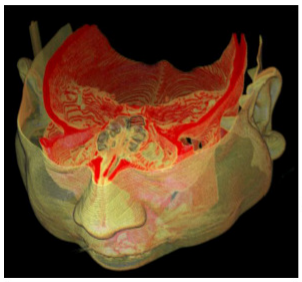


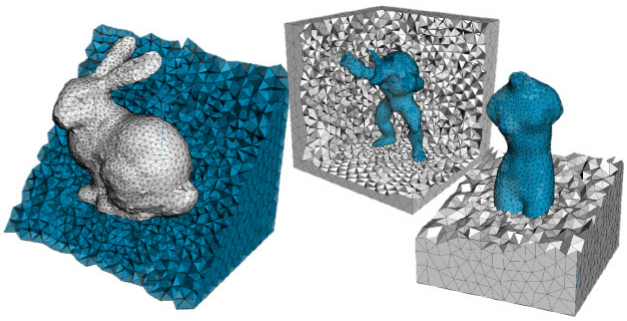
Watertight
Triangular meshes
(modeling software)

Polygon soup
(modeling software)

Representations for modeling & rendering

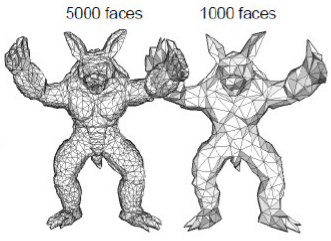
- Volume representations



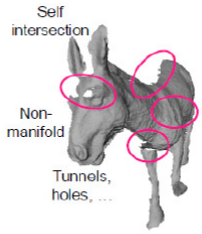


Representations for modeling & rendering

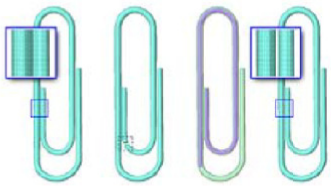
- **Not compact** and **not discriminating**
  - Complex shapes require thousands (millions) of polygons
- **Very sensitive to noise, tessellation, and resolution**
- **Not suitable for comparing shapes**



Resolution



Geometric noise



Topological noise

Shape comparison pipeline

- **Alignment and normalization** (normalize the 3D model)
  - Discard transformations such as translation, scale, and rotations
- Computing **shape descriptors**
  - Compact representations that capture
    - The main features of the shapes
    - Features that discriminate this shape from others
- **Normalization** again (normalize the descriptors)
  - Make the descriptors invariant to various transformations
- …

Shape comparison pipeline

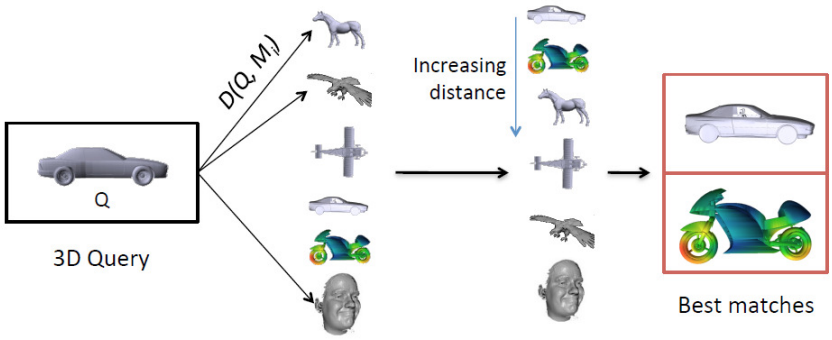
- …
- **Dimensionality reduction**
  - Reduce dimensionality of the descriptors while keeping / improving their discrimination power
- **Machine learning** (supervised / non-supervised)
  - Capture the high-level semantic features

What makes a good descriptor ?

- **Compactness**
  - The smaller the size the better is the representation
- **Discriminative**
  - Distance between similar shapes should be small
  - Distance between dissimilar shapes should be large
- **Robustness**
  - Resolution, geometric and topological noise
- **Invariance**
  - Rigid transformations (translation, rotation, and scale)
  - Non-rigid transformations (bending, elasticity)
  - Representation (point cloud, depth map, polygon soup, …)

Example: 3D retrieval

- Compute the distance from the query to each database model
- Sort the database models by increasing proximity
- Return the closest matches



What we cover in this lecture

- In this lecture, we will address some basic **shape analysis technique** for 3D object classification / recognition / retrieval
- In particular, we will focus on the following descriptors
  - Shape index and curvedness
  - Shape distributions
  - mesh-LBP
  - mesh-DOG / mesh-HOG
  - [mesh-SIFT, optional]

CURVATURE BASED 3D SHAPE DESCRIPTORS

Shape index

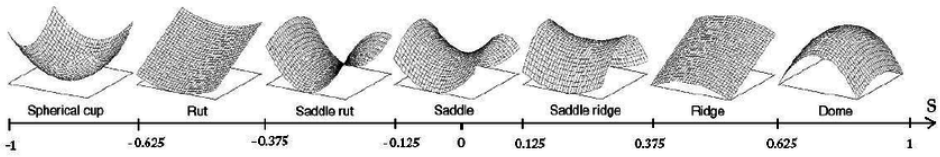
- Koenderink and van Doorn defined a curvature representation from basic curvature measures [1]
- Their approach decouples the **shape** and the **magnitude** of the **curvedness**
- In terms of relative curvature, the surface remains invariant under changes in scale
- They defined a ***shape index*** S, which is a number in the range [-1,1], which describes **local surface topology** in terms of the principal curvatures k_1 and k_2

$$S = \frac{2}{\pi}\tan^{-1}\left(\frac{k_1 + k_2}{k_1 - k_2}\right) \quad k_1 \geq k_2$$

The **shape Index** captures the **shape** of the **curvedness**

Shape index

- The shape index (SI) covers all shapes except for the planar shape, which has an indeterminate shape index ($k_1 = k_2 = 0$)
- The SI provides a continuous gradation between shapes, such as **concave** shapes ($-1 < S < -1/2$), **hyperboloid** shapes ($-1/2 < S < 1/2$) and **convex** shapes ($1/2 < S < 1$)
- The figure below shows the range of SI values and the type of curvature which they represent



Curvedness

- Beside the shape index, Koenderink introduced the positive value *C* for describing the **magnitude** of the **curvedness** at a point

$$C = \sqrt{\frac{k_1^2 + k_2^2}{2}}$$

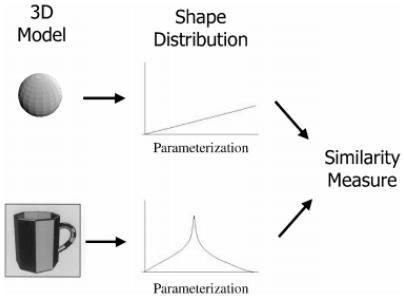
C captures the **magnitude** of the **curvedness**

- It is a measure of how highly or gently curved a point is
- At a point that has no curvedness the value becomes zero
- Therefore, this variable can be used to recognize a planar surface

SHAPE DISTRIBUTIONS

Shape distributions

- **Shape distributions** [2] are a method for computing 3D shape signatures and dissimilarity measures for arbitrary objects described by possibly degenerate 3D polygonal models
- The key idea is to represent the **signature** of an object as a **shape distribution sampled** from a ***shape function*** measuring **global geometric properties** of the object



Shape distributions facilitate shape matching because they represent 3D models as functions with a common parameterization

Motivation

- The primary motivation for this approach is that the **shape matching problem** is reduced to the **comparison of two probability distributions**, which is a relatively simple problem when compared to the more difficult problems encountered by traditional shape matching methods, such as **pose registration**, **parameterization**, **feature correspondence**, and **model fitting**
- The challenges of this approach are to **select discriminating shape functions**, to develop efficient methods for **sampling** them, and to robustly compute the dissimilarity of probability distributions

Overview of the approach

- The basic idea of the approach is to represent the **shape signature** for a 3D model as a **probability distribution sampled** from a **shape function** measuring geometric properties of the 3D model
- This generalization of geometric histograms is called a **shape distribution**

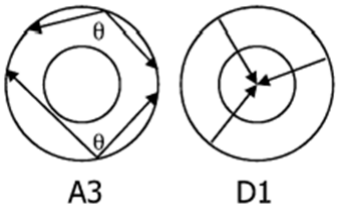
Overview of the approach

- One example shape distribution, called **D2**, represents the **distribution** of **Euclidean distances between pairs of randomly selected points on the surface** of a 3D model
  - While the hypothesis is that the distribution describes the overall shape of the represented object, samples from this distribution can be computed quickly and easily
- Once the shape distributions for two objects have been computed, the **dissimilarity** between the objects can be evaluated using any metric that measures distance between distributions (e.g., L_N norm)

Shape functions

- **A3**: Measures the **angle** between **three random points** on the surface of a 3D model
- **D1**: Measures the **distance** between a **fixed point** and one **random point** on the surface. The centroid of the boundary of the model is used as the fixed point

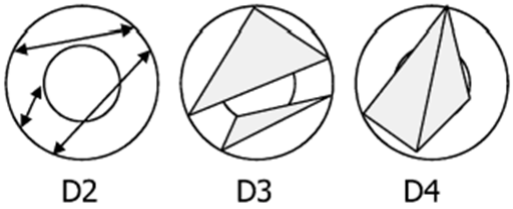
Shape functions based on **angles** (A3) and **lengths** (D1)



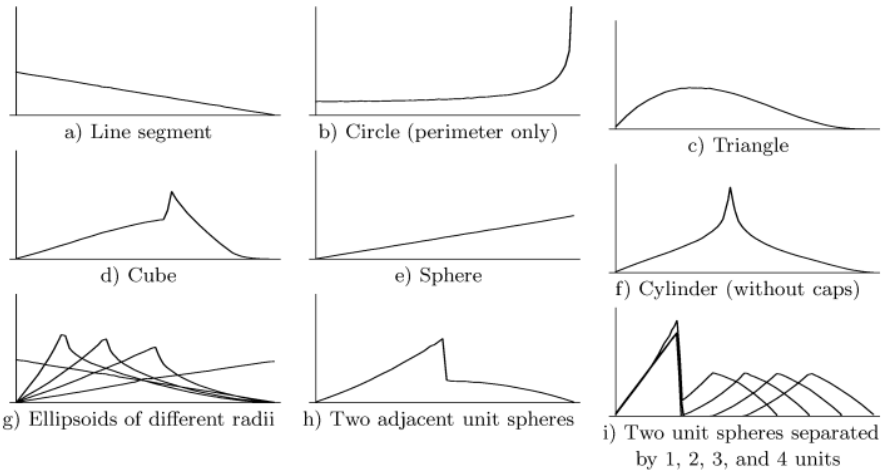
Shape functions

- **D2**: Measures the **distance** between **two random points** on the surface
- **D3**: Measures the square root of the **area** of the **triangle between three random points** on the surface
- **D4**: Measures the cube root of the **volume** of the **tetrahedron between four random points** on the surface

Shape functions based on **lengths** (D2), **areas** (D3), and **volumes** (D4)



Shape functions example



D2 shape distributions. In each plot: the horizontal axis represents distance; the vertical axis represents the probability of that distance between two points on the surface

Shape functions

- These five shape functions were chosen mostly for their **simplicity** and **invariances**
  - They are **quick** to compute, easy to understand, and produce distributions that are **invariant to rigid motions** (translations and rotations)
  - They are **invariant to tessellation** of the 3D polygonal model, since points are selected randomly from the surface
  - They are **insensitive to small perturbations** due to noise, cracks, and insertion/removal of polygons, since sampling is area weighted
  - In addition, the A3 shape function is **invariant to scale**, while the others have to be normalized to enable comparisons
- Finally, the D2, D3, and D4 shape functions provide a nice comparison of 1D, 2D, and 3D geometric measurements

Constructing shape distributions

- Having chosen a **shape function**, the next issue is to compute and store a representation of its **distribution**
- **Analytic calculation** of the distribution is feasible only for certain combinations of shape functions and models (e.g., the D2 function for a sphere or line). **Stochastic methods** are used
  - *N* **samples** are evaluated from the shape distribution and a **histogram** is constructed by counting how many samples fall into each of *B* fixed sized bins
  - From the histogram, a **piecewise linear function** is reconstructed with *V* (<=B) equally spaced vertices, which forms the representation for the shape distribution
- The shape distribution is computed once for each model and stored as a sequence of *V* integers

Issues: sampling density

- One issue is **sampling density**
  - The more samples we take, the more accurately and precisely we can reconstruct the shape distribution
- On the other hand, the time to sample a shape distribution is linearly proportional to the number of samples, so there is an accuracy/time tradeoff in the choice of *N*
- Similarly, a larger number of vertices yields higher resolution distributions, while increasing the storage and comparison costs of the shape signature
  - Empirically, it has been found that using $N = 1024$ samples, $B = 1024$ bins, and $V = 64$ vertices yields shape distributions with **low enough variance** and **high enough resolution** to be useful

Issues: sample generation

- A second issue is **sample generation**. Although it would be simplest to sample vertices of the 3D model directly, the resulting shape distributions would be biased and sensitive to changes in tessellation
- Instead, shape functions are **sampled** from **random points on the surface** of a 3D model

Issues: sample generation

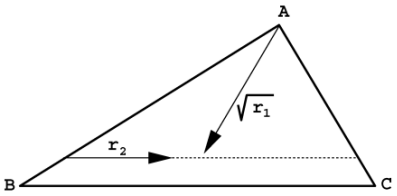
- The method for **generating unbiased random points** with respect to the **surface area** of a **polygonal model** proceeds as follows
  - First, it iterates through all **polygons**, splitting them into **triangles** as necessary
  - Then, for each **triangle**, its **area** is computed and stored in an array along with the cumulative area of triangles visited so far
  - Next, a triangle is selected with probability proportional to its area by generating a random number between 0 and the total cumulative area and performing a binary search on the array of cumulative areas

Issues: sample generation

- For each **selected triangle** with vertices (*A, B, C*), a point on its surface is constructed by **generating two random numbers**, r_1 and r_2, between 0 and 1, and evaluating the following equation

$$P = (1 - \sqrt{r_1})\,A + \sqrt{r_1}(1 - r_2)\,B + \sqrt{r_1} r_2\,C$$

Intuitively, `sqrt`(r_1) sets the percentage from vertex *A* to the opposing edge, while r_2 represents the percentage along that edge. Taking the square-root of r_1 gives a uniform random point with respect to surface area



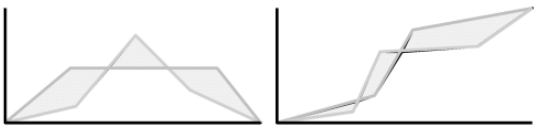
Sampling a **random point** in a triangle

Comparing shape distributions

- Having constructed the **shape distributions** for two 3D models, we are left with the task of **comparing them** to produce a **dissimilarity measure**
- Eight dissimilarity measures have been experimented with
  - The first two are the χ^2 statistic and the *Bhattacharyya* distance, which are commonly used for comparing binned data
  - The remaining six are *Minkowski* L_N norms of the **probability density functions** (**pdfs**) and **cumulative distribution functions** (**cdfs**) for $N = 1, 2, \infty$

Comparing shape distributions

- The dissimilarity measures are computed as follows, assuming f and g represent **pdfs** for two models, while $\hat{f}$ and $\hat{g}$ represent the corresponding **cdfs** that is, $\hat{f}(x) = \int_{-\infty}^{x} f$
- χ^2: $D(f,g) = \int \frac{(f-g)^2}{f+g}$.
- **Bhattacharyya:** $D(f,g) = 1 - \int \sqrt{fg}$.
- **PDF** L_N: Minkowski L_N norm of the pdf: $D(f,g) = (\int |f-g|^N)^{1/N}$.
- **CDF** L_N: Minkowski L_N norm of the cdf: $D(f,g) = (\int |\hat{f}-\hat{g}|^N)^{1/N}$.



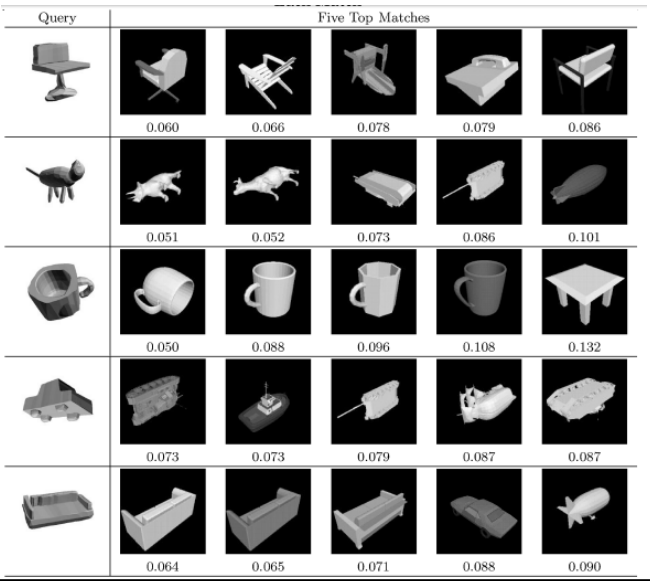
PDF CDF

Comparison of **shape distributions** can be performed by computing the L_N norm of the **PDF** (left) or **CDF**(right) representations.
Areas of the gray regions in each plot represent the corresponding L_1 norm

Comparing shape distributions

- Since each shape distribution is represented as a piecewise linear function, analytic computation of these norms can be done efficiently in time proportional to the number of vertices used to store the distributions

Applications: 3D retrieval example



3D **query models** (left column) and the five top matches for each query returned by a web-based search engine.
D2 dissimilarity measures are shown below each match

LOCAL SURFACE DESCRIPTORS: MESH-LBP

2D LBP overview

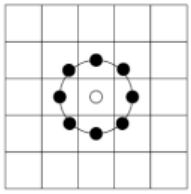
- In its original definition, the LBP operator [3] assigns labels to image pixels by
  - First, thresholding the 3 × 3 neighborhood of each pixel with the center value (i.e., each pixel in the neighborhood is regarded as 1 if its value is greater or equal to the central value, 0 otherwise)
  - Then, considering the sequence of 0/1 in the pixel neighborhood as a binary number according to a positional coding convention

2D LBP overview

- The idea of 2D LBP is shown below, where the upper left pixel in the neighborhood is regarded as the most significant bit in the final code

92	59	33
15	66	21
79	37	71

10001010
(binary value 138)



(a) (b)

(a) Computation of the basic LBP code from the 3 × 3 neighborhood of a central pixel. Each pixel, starting from the upper-left corner is compared with the central pixel to produce 1 if its value is greater or equal, 0 otherwise. The result is an 8-bit binary code; (b) Example of a central pixel with a circular neighborhood of a given radius

2D LBP overview

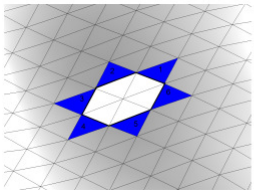
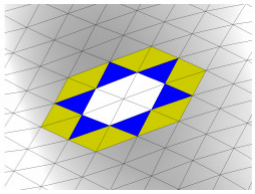
- This eight bits number encodes the mutual relationship between the gray levels of the central pixel and its neighboring pixels
- The histogram of the numbers obtained in such a way can then be used as a **texture descriptor**
- This operator is distinguished by its **simplicity** and its **invariance** to **monotonic gray-level transformations**
- An extended LBP version that can operate on circular neighborhood of different radii (see case (b) in the previous slide), also allowing sub-pixel alterations was proposed in [4]
- Several other 2D LBP extensions have been proposed in literature mainly differing in the choice of pixels to compare

mesh-LBP

- **Mesh-LBP** extends the LBP concept to **2D mesh manifolds** [5]
- The construction of LBP-like patterns on a mesh, first requires a scheme for constructing **rings of facets around a central one** and for traversing them in an **ordered fashion**
- Let $S = \{V, F\}$ be the triangular mesh representation of an open or closed surface, where V and F are, respectively, the sets of vertices and facets of the mesh
- Let us start by considering the general case of a **convex contour** on the mesh, which we assume regular, i.e., each vertex has a valence of six (actually the framework can also cope with meshes that do not comply with this ideal case)

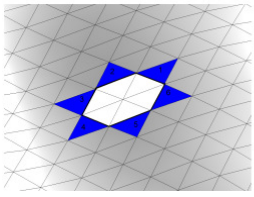
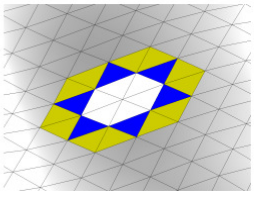
Ordered ring facets

- Consider the facets that have an edge on the convex contour (Figure (a))
  - These facets are called ***Fout*** facets, as they seem pointing outside the contour
- Let us consider also the set of facets that are one-to-one adjacent to the *Fout* facets and which are located inside the convex contour
  - Each facet in this set, called ***Fin***, shares with its corresponding *Fout* facet an edge located on the convex contour

(a)  (b) 

Ordered ring facets

- Let us assume that the *Fout* facets are initially **ordered** in a **circular fashion** across the contour
- Given that initial arrangement, the gap between each pair of consecutive *Fout* facets is bridged, that is the sequence of adjacent facets, located between the two consecutive *Fout* facets and which share their common vertex (the vertex on the contour) is extracted
- These facets are called ***Fgap*** facets (see Figure (b))

(a)  (b) 

47

Ordered ring facets

- The "**Bridge**" procedure reported here in pseudocode is used to compute the **Fgap** facets

Algorithm 1 – Bridge

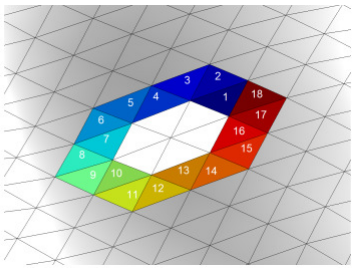
Input: $fout_i$, $fout_{i+1}$ two consecutive $Fout$ facets sharing a vertex; fin_i facet which shares an edge with $fout_i$
Output: $Fgap_i$ set of consecutive $fgap$ facets bridging the gap between $fout_i$ and $fout_{i+1}$

procedure BRIDGE($fout_i, fout_{i+1}, fin_i$)
  $Fgap_i$ = [ ]
  $v \leftarrow$ vertex shared by $\langle fout_i, fout_{i+1} \rangle$
  $gf \leftarrow$ facet adjacent to $fout_i$, different from fin_i and containing v
  $prev \leftarrow fout_i$
  while $gf \neq fout_{i+1}$ **do**
    append gf to $Fgap_i$
    $new\_gf \leftarrow$ facet adjacent to gf, different from $prev$ and containing v
    $prev \leftarrow gf$
    $gf \leftarrow new\_gf$
  end while
  return $Fgap_i$
end procedure

Ordered ring facets

- By iterating the process of bridging the gap between two consecutive *Fout* facets with the *Fgap* facets results in a ring of facets that are ordered in a circular fashion (see Figure (c))
- The resulting arrangement of the ring facets inherits the same direction (clock-wise or anti-clockwise) of the initial sequence of *Fout* facets

(c)



Ordered ring facets

- The "**GetRing**" procedure constructs a ring by iterative calls of the "Bridge" procedure
- The obtained ordered ring is called **Ordered Ring Facets** (**ORF**)

Algorithm 2 – GetRing

Input: $Fout$, set of n ordered facets, $fout_1$, $fout_2$, ..., $fout_n$, lying on a convex contour; Fin, set of n ordered facets, $fin_1, fin_2, \ldots, fin_n$, one-to-one adjacent to the $Fout$ facets and located inside the region delimited by the convex contour (depending on the contour, Fin might include duplicates)
Output: $Ring$, ring of ordered facets

procedure GETRING($Fout, Fin$)
  $Ring$ = [ ]
  for all $\langle fout_i, fout_{i\%n+1}\rangle$, $i \leftarrow 1, \ldots, n$ **do**
    append $fout_i$ to $Ring$
    $Fgap_i \leftarrow$ BRIDGE($fout_i, fout_{i\%n+1}, fin_i$)
    append $Fgap_i$ to $Ring$
  end for
  return $Ring$
end procedure

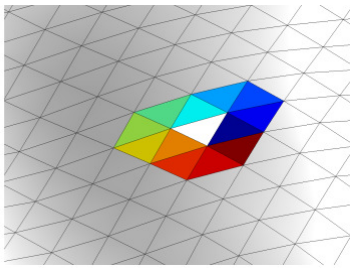
Ordered ring facets

- In the above discussion, we referred to the general case where the ORF is constructed around a convex contour
- Actually, the usual case is constituted by an initial seed formed by an **individual facet** (**central facet**), whose **three edges represent the initial convex contour**
- This case corresponds to the most common case where an ordered ring is constructed around the facets of a mesh surface

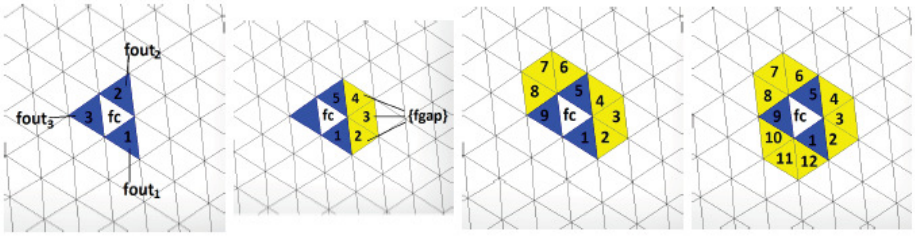
Ordered ring facets

- In this particular case, the *Fout* set includes the three facets adjacent to the central one, and the obtained ring is composed of 12 ordered facets (i.e., the three *Fout* facets, plus the nine *Fgap* facets bridging the gap between consecutive *Fout* facets), as shown in Figure (d)

(d)



Ordered ring facets



Ordered ring construction: From the initial *Fout* facets formed by the three ordered facets $fout_1$, $fout_2$, and $fout_3$ that are adjacent to the central facet fc, a sequence of *Fgap* facets located between each pair $<fout_1, fout_2>$, $<fout_2, fout_3>$, and $<fout_3, fout_1>$ are extracted. The *Fgap* facets have exactly one vertex on the initial three-edges contour of the central facet fc, and they are dubbed so because they look like filling the gap between the *Fout* facets. This procedure obtains a ring of facets ordered in a circular fashion around the central facet fc

mesh-LBP

- Let *h(f)* : *S* → R be a **scalar function** defined on the mesh *S* (e.g., **photometric data** or **curvature**)
- The **circular ordering of the facets obtained with ORF** allows us to derive a **binary pattern** (i.e., sequence of 0 and 1 digits) from it, and thus to compute a local binary operator in the same way as in the standard 2D LBP
- The basic mesh-LBP operator at a central facet f_c is defined by thresholding its ordered ring neighborhood constituted by the 12 facets in the ORF

$$meshLBP(f_c) = \sum_{k=0}^{11} s(h(f_k) - h(f_c)) \cdot \alpha(k)$$

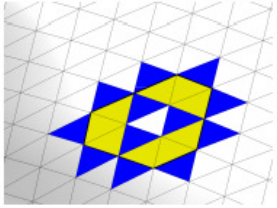
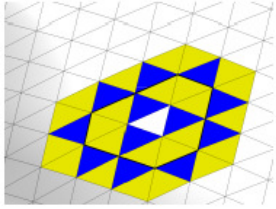
$$s(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases},$$

mesh-LBP

- $\alpha(k)$ is a weighting function that admit different definitions that permit us to obtain different binary patterns, and thus different mesh-LBP values can be derived from the central facet and its ring neighborhood
- For example
  - With $\alpha(k) = 2^k$ the basic LBP operator firstly suggested by Ojala et al. [3] is obtained
  - For $\alpha(k) = 1$, the **sum of the digits of the pattern is computed** (i.e., the number of digits equal to 1)
- For the present discussion it is not necessary to detail the particular scalar function *h(f)*, whose values are computed on the mesh facets

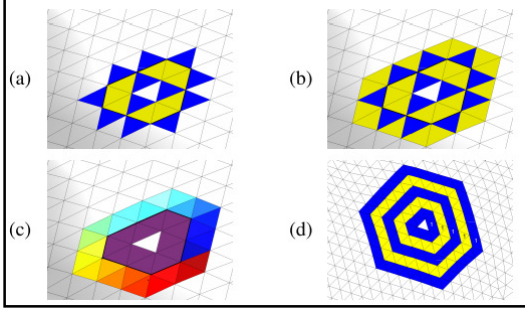
Multi-resolution mesh-LBP

- The mesh-LBP is extended to a multi-resolution framework by deriving a **sequence of concentric rings**, which preserve the ordering property
- From the first ring, the sequence of facets that are one-to-one adjacent to the *Fgap* facets are extracted (Figure (a))
- This sequence, which inherits the order property of the *Fgap* facets, constitutes the set of *Fout* facets for the subsequent ring

(a)  (b) 

Multi-resolution mesh-LBP

- By filling the gap between each two consecutive facets of this sequence (Figure (b)), a new ring, which exhibits the same ordered structure of its predecessor is obtained (Figure (c))
- By iterating this procedure, a sequence of concentric ordered rings is built, which represents the primitive entity for computing multi-resolution mesh-LBP (Figure (d))



Construction of multi-resolution mesh-LBP: (a) Extraction of the next set of *Fout* facets, as the facets adjacent to *Fgap* which are not part of the current ring; (b) Extracting the *Fgap* facets; (c) The second ordered ring extracted; (d) Five concentric ordered rings. Notice that the first facet of each ring (marked by 1) is located at the same relative position

Multi-resolution mesh-LBP

- Procedure used for computing the multi-ring structure

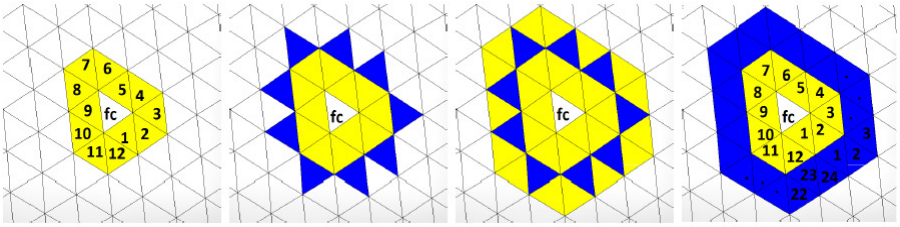
Algorithm 3 – MultiRing

Input: $Fout\_root$, initial set of ordered $Fout$ facets; Fin_root, initial set of ordered Fin facets one-to-one adjacent to the $Fout$ facets; Nr, number of rings to be constructed around Fin_root

Output: $Rings$, set of Nr rings of ordered facets constructed around Fin_root

```
procedure MULTIRING(Fout_root, Fin_root, Nr)
    Rings ← [ ]
    Fout ← Fout_root
    Fin ← Fin_root
    for i ← 1, Nr do
        (Ring, NewFout, Fgap) ← GETRING(Fout, Fin)
        append Ring to Rings
        Fout ← NewFout
        Fin ← Fgap
    end for
    return Rings
end procedure
```

Multi-resolution mesh-LBP



By iterating the ring construction procedure, using as new set of *Fout* facets the sequence of facets that share an edge on the outer contour of the current ring, a **sequence of rings of ordered facets** can be generated

Multi-resolution mesh-LBP

- In this case, the "GetRing" procedure is slightly modified, so that it also returns the set of *Fgap* facets of the current ring and the set of *Fout* facets of the subsequent ring (indicated as *NewFout*)
- It is worth mentioning that, when the regularity assumption for the mesh is satisfied, the number of facets ν across the rings evolves according to the following **arithmetic progression** from ring i to ring i+1: $\nu_{i+1} = \nu_i + 12$
- This can be intuitively seen referring to the previous Figure: the 1st-ring comprises 12 facets (3 *Fout* plus 9 *Fgap*); 24 facets are included in the 2nd ring (i.e., 9 *Fout* plus 15 *Fgap*); 36 facets in the third ring, and so on

Multi-resolution mesh-LBP

- Given a multi-ring constructed around a central facet f_c, a multi-resolution mesh-LBP operator is derived as follows

$$meshLBP_m^r(f_c) = \sum_{k=0}^{m-1} s(h(f_k^r) - h(f_c)) \cdot \alpha(k),$$

  where *r* is the ring number, and *m* is the number of facets uniformly spaced on the ring
- The parameters *r* and *m* control, respectively, the radial resolution and the azimuthal quantization of the operator
- In principle, any predefined number of samples per ring can be used (a typical number is *m* = 12)

Uniform patterns

- By studying the statistics of the number of bitwise 0-1 transitions in the binary patterns, Ojala et al. [4] noticed that the majority of the patterns in textured 2D images have a **number of transitions** *U* equal at most to 2
- These patterns are called "**uniform**"
- There is evidence of the existence of a "uniformity" aspect of the mesh-LBP computed on triangular mesh manifolds
- Used surface descriptors (*f* function)
  - *Mean curvature* (H)
  - *Gaussian curvature* (K)
  - *Curvedness* (C)
  - *Angle between facets normal* (D)

Uniform patterns

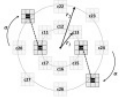
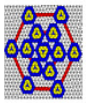
- For each of these functions, the number of transitions *U* in the binary patterns computed by using the mesh-LBP operator is evaluated, across six levels of spatial resolution (*r* from 1 to 6), and using 12 samples for the azimuthal quantization (*m* = 12 at each *r*)
- Based on the obtained results, considering an azimuthal quantization of *m* = 12, that is 4096 possible patterns, we define the set of **uniform patterns** as the set including all **binary patterns** for which *U* is at most equal to 4
- This set contains exactly 1123 patterns against 2973 for the non-uniform patterns

Uniform patterns

- Following the same partition scheme of [4], where all the non-uniform patterns are grouped into a single label, whereas a separate label is assigned to each non-uniform pattern, the number of labels (or classes) is reduced to 1124 for mesh-LBP
  - Notably, this partition is used for the mesh-LBP operator involving $\alpha(k) = \alpha_2(k) = 2^k$
  - For $\alpha_1(k) = 1$ the distinction into uniform/non-uniform patterns does not make too much sense since the number of patterns is already small (13 patterns exactly)

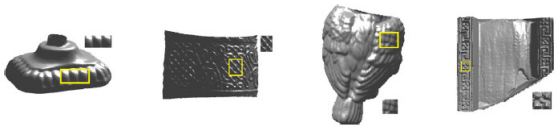
Mesh-LBP variants

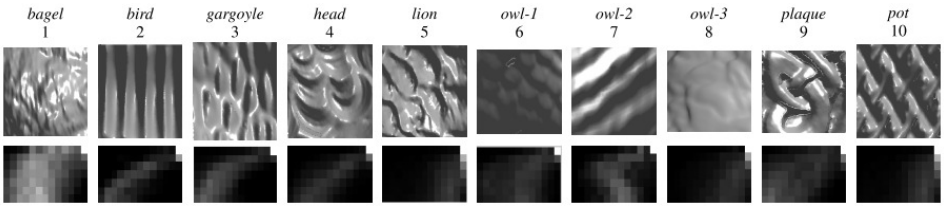
The mesh-LBP framework allows for the inclusion of many **variants**, which account for the way neighbor facets to the central one are selected and compared

Class of variation	LBP variant	2D-LBP pattern	mesh-LBP pattern
Browsing path	(a) Local-line [43]		
	(b) Archimedian Spiral [44]		
Contour / initial seed	(c) Elongated [18]		
Comparison scheme	(d) Center-symmetric [29]		
Structural element	(e) Three-Patch [10]		
	(f) Four-Patch [10]		

Applications: 3D texture retrieval

Some 3D texture samples from the ten classes

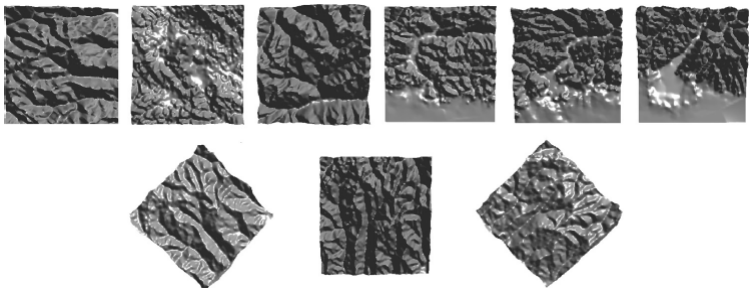




The corresponding histograms obtained with the angle between facets normal and the α_1 weighting function using 7 rings and 12 samples per ring (i.e., histograms with 7 rows and 13 columns). Each histogram bin cumulates the frequency of a mesh-LBP pattern computed for all the facets of a sample surface (histograms are represented as gray-level images, where lighter pixels correspond to histogram bins with higher values)

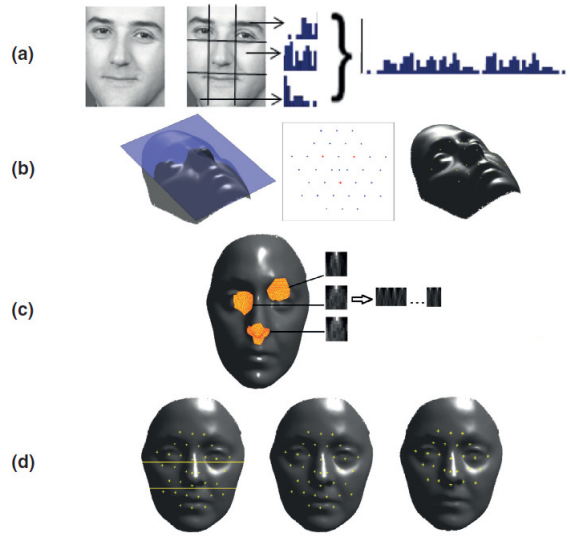
Applications: 3D terrain retrieval

- Given a 3D terrain query representing a specific area, find its corresponding match in a gallery of 3D terrain models
- Terrain models are originally **digital elevation models** (DEM) converted into mesh models



Applications: 3D face recognition

- 3D face recognition based on mesh-LBP computed on **surface photometric** and **curvature** data



MESH DOG / HOG

Motivation

- Mesh-DOG / HOG is a computational framework that allows estimating **interest points** and **local descriptors** of a **scalar function** defined over a mesh manifold [10,11]
  - It is inspired by Scale Invariant Feature Transform (SIFT) [7]
- This requires definition of several operators that can handle an irregular domain, including the **gradient operator** and the **first**- and **second order directional derivative** operators

Differential mesh processing

- Let $\mathcal{M}$ be a 2D **closed manifold** (i.e., compact and without boundaries) embedded in R^3 and let M be a **discrete mesh** representation of $\mathcal{M}$ composed of vertices on M and of convex polygons, i.e., facets
- M can be viewed as a graph $M(V,E)$, where $V = \{v_i\}^N_{i=1}$ is the set of **mesh vertices** and $E = \{e_{ij}\}$ is the set of **edges** between adjacent vertices
- We associate a 3D point $\mathbf{v}_i \in \mathrm{R}^3$ with each **mesh vertex** v_i

Gradient

- Let f: $\mathcal{M} \rightarrow \mathrm{R}$ be a smooth real-valued function defined on $\mathcal{M}$, e.g., **photometric data** or **curvature** (*mean*, *Gaussian*)
  - Note that differently from mesh-SIFT (discussed later in this Lecture) here the **operations are applied to a function defined on the mesh** rather than to the mesh itself
- In order to estimate the ***gradient*** $\nabla_{\mathcal{M}} f(\mathbf{v})$ of f at point $\mathbf{v}$, we consider the **first order Taylor expansion** approximating f at a manifold point $\mathbf{v}_i$ in a neighborhood of point $\mathbf{v}_j$

$$f(\mathbf{v}_i) \approx f(\mathbf{v}_j) + \nabla_{\mathcal{M}} f(\mathbf{v}_i)^T (\mathbf{v}_i - \mathbf{v}_j) \qquad (1)$$

where the gradient belongs, by definition, to the **tangent plane** of $\mathcal{M}$ at v_i

Gradient

- In the discrete case one can therefore write

$$\nabla_M f(\mathbf{v}_i)^T(\mathbf{v}_i - \mathbf{v}_j) \approx f(\mathbf{v}_i) - f(\mathbf{v}_j) \quad (2)$$

  where $\nabla_M f(\mathbf{v})$ denotes the ***discrete gradient*** of f at $\mathbf{v}$
- This expression can be used to estimate the discrete gradient at any mesh vertex $\mathbf{v}_i$ through an **error minimization criterion**
- This adopts the **least square gradient construction** seeking the 3D vector $\mathbf{g}$ that minimizes the criterion

$$\nabla_M f(\mathbf{v}_i) = \arg\min_{\mathbf{g}} \left\{ \sum_{v_j \sim v_i} w_{ij} (f(\mathbf{v}_i) - f(\mathbf{v}_j) - \mathbf{g}^T(\mathbf{v}_i - \mathbf{v}_j))^2 \right\} \quad (3)$$

  where $v_j \sim v_i$ means that $v_j \in N(\mathbf{v}_i)$, i.e., the neighborhood of $\mathbf{v}_i$ considered in the estimation, and where the weights w_{ij} balance the contributions of the neighboring vertices

Gradient

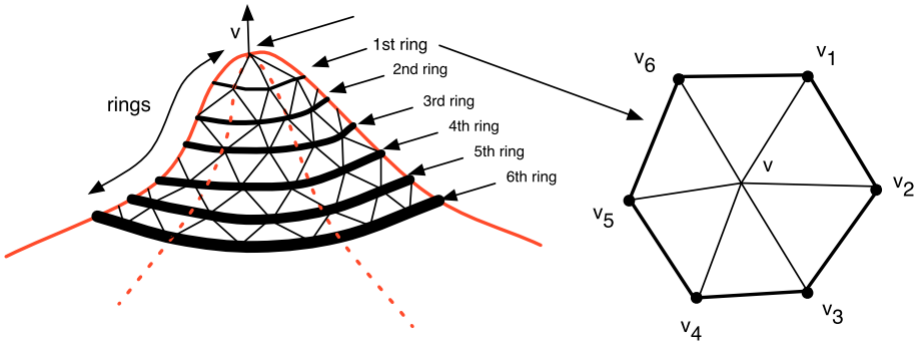
$N(\mathbf{v}_i)$ and w_{ij} are chosen as follows

- $N(\mathbf{v}_i)$ is usually the **first ring** of vertices around $\mathbf{v}_i$
  - In order to make it more robust to non-uniform sampling, $N(\mathbf{v}_i)$ can be defined as the set of vertices $\mathbf{v}_j \in M$ residing within a **geodesic ball** centered in $\mathbf{v}_i$ of radius r

$$N(\mathbf{v}_i) = \{\mathbf{v}_j \mid d_g(\mathbf{v}_i, \mathbf{v}_j) < r\}$$

    where $d_g(\mathbf{v}_i, \mathbf{v}_j)$ represents the **geodesic distance** between $\mathbf{v}_i$ and $\mathbf{v}_j$

Gradient



A vertex *v* and its rings (left), and the first ring of *v* (right)

Gradient

- The **weight function** w_{ij} can be uniform or it can vary with respect to, e.g., areas or inverse distances in the neighborhood of $\mathbf{v}_i$
- In any case, it is shown weighted gradient estimations based on **inverse distances** significantly improve over unweighted estimations
- The **weight function** is defined as a **zero centered Gaussian function**

$$w_{ij} = G_\sigma(d_g(\mathbf{v}_i, \mathbf{v}_j)) = \exp(-d_g^2(\mathbf{v}_i, \mathbf{v}_j)/2\sigma^2)$$

where $d_g(\mathbf{v}_i,\mathbf{v}_j)$ is the **geodesic distance** between $\mathbf{v}_i$ and $\mathbf{v}_j$

Gradient

- Vector $\mathbf{g}$ in Eq.(3) can be advantageously constrained to belong to the tangent plane of M at $\mathbf{v}_i$, whenever the **normal unit vector** $\mathbf{n}_i$ to this tangent plane is known

$$\nabla_M f(\mathbf{v}_i) = \arg\min_{\mathbf{g}} \left\{ \sum_{v_j \sim v_i} w_{ij} (f(\mathbf{v}_i) - f(\mathbf{v}_j) - \mathbf{g}^T(\mathbf{v}_i - \mathbf{v}_j))^2 + \lambda_i (\mathbf{g}^T \mathbf{n}_i)^2 \right\} \quad (4)$$

  where the positive scalar λ_i is chosen such that the **tangent plane constraint** is emphasized $\lambda_i = \sum_{v_j \sim v_i} w_{ij}$

- The above constrained minimization is a **linear least-squares problem** that is efficiently solved using standard **matrix factorization methods**, such as **Singular Value Decomposition** (SVD)

Directional derivatives

- The ***directional derivative*** $D_{\mathbf{a}} f(\mathbf{v})$ of f at $\mathbf{v} \in \mathcal{M}$ along vector $\mathbf{a}$ is then, by definition, the **projection of the gradient vector** $\nabla_M f(\mathbf{v})$ along the direction of $\mathbf{a}$

$$D_{\mathbf{a}} f(\mathbf{v}) = \nabla_{\mathcal{M}} f(\mathbf{v})^T \frac{\mathbf{a}}{\|\mathbf{a}\|} \quad (5)$$

  where $\mathbf{a}$ is a **vector lying** in the **tangent plane** of $\mathcal{M}$ at $\mathbf{v}$

- The discrete directional derivative at $\mathbf{v}_i$ along any direction in the tangent plane at $\mathbf{v}_i$ can therefore be computed using Eq.(5) with the discrete gradient $\nabla_M f(\mathbf{v}_i)$ estimated using Eq.(4)

Directional derivatives

- Let us now consider ***second*** **discrete directional derivatives** along unit vectors $\mathbf{a}$ and $\mathbf{b}$ lying in the tangent plane to the mesh at $\mathbf{v}_i$
- Such a second directional derivative can be written as

$$D_{\mathbf{ab}} f(\mathbf{v}_i) = \nabla_M (\nabla_M f(\mathbf{v}_i)^T \mathbf{b})^T \mathbf{a}$$
$$= D_{\mathbf{a}} (D_{\mathbf{b}} f(\mathbf{v}_i))$$

  which requires an **estimate of the gradient** of the **scalar function** $D_{\mathbf{b}} f(\mathbf{v}_i)$, namely $\nabla D_{\mathbf{b}} f(\mathbf{v}_i)$
- This can be easily obtained by applying the least square criterion of Eq.(4) to the directional derivative function

Directional derivatives

- Similarly, one can estimate $D_{\mathbf{aa}} f(\mathbf{v}_i)$, $D_{\mathbf{bb}} f(\mathbf{v}_i)$, $D_{\mathbf{ab}} f(\mathbf{v}_i)$ and $D_{\mathbf{ba}} f(\mathbf{v}_i)$
- By further assuming that the vectors $\mathbf{a}$ and $\mathbf{b}$ form an **orthonormal basis of the tangent plane** at $\mathbf{v}_i$, one obtains the **Hessian matrix** of f

$$\mathbf{H}_{\mathbf{ab}}(f(\mathbf{v}_i)) = \begin{bmatrix} D_{\mathbf{aa}} f(\mathbf{v}_i) & D_{\mathbf{ab}} f(\mathbf{v}_i) \\ D_{\mathbf{ba}} f(\mathbf{v}_i) & D_{\mathbf{bb}} f(\mathbf{v}_i) \end{bmatrix} \qquad (6)$$

Directional derivatives

- Assuming that the directional derivatives of f are continuous, one should expect the Hessian in Eq.(6) to be **symmetric**, namely that $D_{\mathbf{ab}}f(\mathbf{v}_i) = D_{\mathbf{ba}}f(\mathbf{v}_i)$
- However, in our case, the gradients $\nabla D_{\mathbf{a}}f(\mathbf{v}_i)$ and $\nabla D_{\mathbf{b}}f(\mathbf{v}_i)$ are obtained by numerical optimization of Eq.(4)
- Hence, the Hessian is not guaranteed to be symmetric, and so it is not guaranteed that the two eigenvalues of Eq.(6) are real
- Therefore, the following **real symmetric matrix** is used instead

$$\tilde{\mathbf{H}}_{\mathbf{ab}}(f(\mathbf{v}_i)) = \frac{1}{2}(\mathbf{H}_{\mathbf{ab}}(f(\mathbf{v}_i)) + \mathbf{H}^T_{\mathbf{ab}}(f(\mathbf{v}_i)))$$

which corresponds to the projection of $\mathbf{H}$ onto the linear space of 2×2 symmetric matrices

Convolution

- Finally, using the same notations, the ***normalized convolution*** of the function f with a **Gaussian kernel** G yields

$$F_\sigma(\mathbf{v}_i) = f \otimes G_\sigma(\mathbf{v}_i) = \frac{1}{K_i}\sum_{v_j \sim v_i} f(\mathbf{v}_j)\exp(-d_g^2(\mathbf{v}_i,\mathbf{v}_j)/2\sigma^2) \qquad (7)$$

where G_σ is the Gaussian function previously defined and K_i is a normalization term such that $K_i = \sum_{v_j \sim v_i} G_\sigma(d_g^2(\mathbf{v}_i,\mathbf{v}_j))$

- Using the properties of convolution, one can easily compute the **first- and second-order directional derivatives** of F_σ

$$D_{\mathbf{a}}F_\sigma(\mathbf{v}_i) = D_{\mathbf{a}}f \otimes G_\sigma(\mathbf{v}_i)$$
$$D_{\mathbf{ab}}F_\sigma(\mathbf{v}_i) = D_{\mathbf{ab}}f \otimes G_\sigma(\mathbf{v}_i) \qquad (8)$$

Numerical approximations

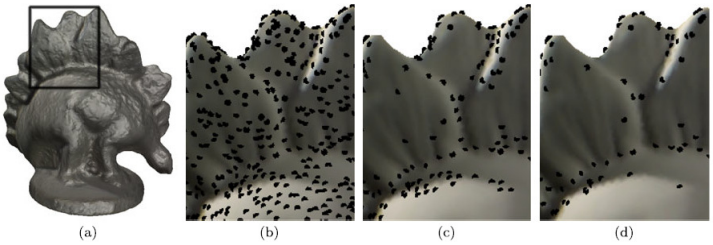
- ***Geodesics*** - The computation of geodesic distances on arbitrarily triangular meshes can be computed by the **fast marching** method or by other approximations
  - In practice, in the interest of computational speed, a local **shortest path approximation** on the edge connectivity graph is used
  - It has been observed experimentally that, for typical meshes, the variations due to the triangulation are minimal

Numerical approximations

- ***Normals*** - At first, a local normal estimation is used on a one ring neighborhood. In order to increase the robustness of the normal estimation, a smoothed version is then computed, using the mean estimate in a small geodesic neighborhood
- ***Curvatures*** **-** Instead of using a classical curvature estimation method [12] that employs only a one ring neighborhood, a more robust method based on a 3 ring neighborhood is used [13]

Feature detection (mesh-DOG)

- Feature detection comprises three steps
  1. The **extrema** of the function's Laplacian are found across scales using a one-ring neighborhood. The **Laplacian** is approximated with the standard **difference of Gaussian** (DOG) operator
  2. The extrema thus detected are **thresholded**
  3. The unstable extrema are eliminated, only retaining the features exhibiting some degree of **cornerness**



Feature detection shown with **photometric data**. (**a**) Original mesh; (**b**) Scale-space extrema; (**c**) Thresholding; (**d**) Corner detection

85

Scale-space construction

- A **scale-space** representation of any **scalar function** f defined on a mesh is considered, built by **progressive convolutions** over f
- The scale-space is built over s = 3 **octaves**, covering each octave in c = 6 **steps**
- This is accomplished by **progressive convolutions** with a **Gaussian kernel** (see Eq.(7)) of the original scalar function

$$F_0 = f$$
$$F_t = F_{t-1} \otimes G_{\sigma(t)} \qquad (9)$$

with $t = \{1, 2, \ldots, s \cdot c\}$

Scale-space construction

- The standard deviation parameter $\sigma(t)$ of the Gaussian at iteration t is chosen based on the formula

$$\sigma(t) = 2^{\frac{1}{c-2}\left\lceil \frac{t}{c} \right\rceil} e_{avg}$$

  where e_{avg} represents the **average edge length**
- As it can be observed
  - The value of $\sigma(t)$ only changes when t starts spanning a **new octave**
  - $\sigma(t)$ remains unchanged for the next c iterations, while t covers the current octave, thanks to the term $\left\lceil \frac{t}{c} \right\rceil$
  - This behavior emulates in spirit the 2D grid down sampling, introduced for SIFT, but without modifying the mesh geometry, which can be an expensive operation, in the case of non-uniformly sampled meshes

Scale-space construction

- An important observation is that, when building the scale space of a scalar function defined over the mesh, the **mesh geometry does not change**
  - This contrasts to other approaches that construct the scale-space by generating meshes with different samplings, thus requiring further mesh processing and simplification
- The **difference of Gaussian operator** (**DOG**) is then used as an approximation of the **Laplacian operator**, built by subtracting adjacent convolved functions

$$L_t = F_t - F_{t-1} \qquad (10)$$

Note: Laplacian of Gaussian and its approximation as difference of Gaussians

- Marr and Hildreth defined an operator for image edge detector called **Laplacian of Gaussian** (LoG) that combines the effect of the **Gaussian filter** (to smooth the gray levels) with that of the **Laplacian operator** (to enhance the edges)
- A good approximation of the Marr-Hildreth's filter is obtained by using an operator based on the difference between two Gaussians with different values of σ
- Marr and Hildreth found that a ratio of σ_2/σ_1=1.6 obtains the best approximation to the LoG filter

Note on Laplacian and DoG

- The **Laplacian** is a 2-D isotropic measure of the 2nd spatial derivative of an image
- The Laplacian of an image highlights regions of **rapid intensity change** and is therefore often used for **edge detection** (see zero crossing edge detectors)
- The Laplacian is often applied to an image that has first been **smoothed** with something approximating a **Gaussian smoothing filter** in order to reduce its sensitivity to noise
- Since the input image is represented as a set of discrete pixels, a **discrete convolution kernel** is used to approximate the second derivatives in the definition of the Laplacian
  - Using these kernels, the Laplacian can be calculated using standard convolution methods

Note on Laplacian and DoG

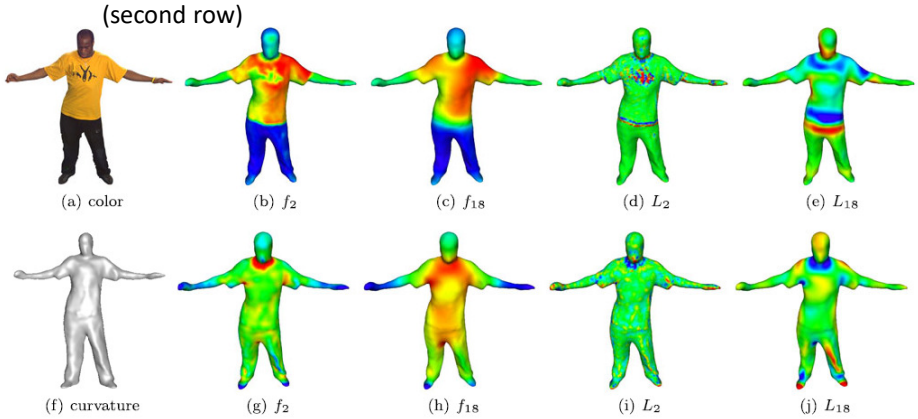
- Because these **kernels** are approximating a **second derivative measurement** on the image, they are very **sensitive to noise**
- To counter this, the image is often **Gaussian smoothed** before applying the Laplacian filter
- This pre-processing step reduces the high frequency noise components prior to the differentiation step
- Since the **convolution operation is associative**, we can **convolve** the **Gaussian smoothing filter with** the **Laplacian filter** first of all, and then **convolve this hybrid filter with the image** to achieve the required result

Note on Laplacian and DoG

- Doing things this way has two advantages
  - Since both the Gaussian and the Laplacian kernels are usually much smaller than the image, this method usually requires far fewer arithmetic operations
  - The LoG (`**Laplacian of Gaussian**') kernel can be precalculated in advance so only one convolution needs to be performed at run-time on the image
- It is possible to **approximate the LoG** filter with a filter that is just the **difference of two differently sized Gaussians**
  - Such a filter is known as a ***DoG* filter** (short for `**Difference of Gaussians**')
- It has been also suggested that LoG filters (actually DoG filters) are important in biological visual processing

Scale-space construction

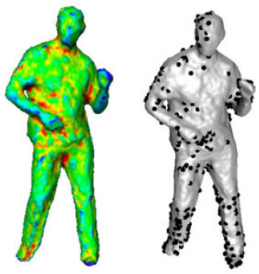
- An example is given below
  - The features shown are: color intensity (first row), and mean curvature (second row)



Feature detection

- **Feature points** represent a subset of all vertices that can be detected with high **repeatability**
  - Using **local gradient information** is one way to detect a repeatable feature
- The **feature points** are selected as the **local extrema** over one ring neighborhoods, in the current and in the adjacent scales

photometric data and associated points of interest



mean surface curvature and the detected features

Feature detection: thresholding

- From the **extrema of the scale space**, only the top β = 5% of the maximum number of vertices are being considered, sorted by magnitude
  - A percentage value, versus a hard value threshold is used in order to keep the detector flexible, no matter which scalar function is being considered, i.e., color intensity or mean curvature, without the need for normalization. However, when the threshold response is known a priori for a particular scalar function it can be easily used instead
- Additionally, in order to eliminate more non-stable responses, only the features that exhibit **corner characteristics** are retained
- This can be done as in SIFT by examining the **eigenvalues** of the 2 × 2 **Hessian matrix of second directional derivatives** of the difference of Gaussian operator

Feature detection: cornerness

- Let us consider the DOG operator L_i , defined in Eq.(10) and the *symmetric* **Hessian matrix** approximation $\tilde{\mathbf{H}}_{\mathbf{xy}}(L_i(\mathbf{v}_i))$, where $(\mathbf{x},\mathbf{y})$ is a pair of orthonormal vectors lying in the tangent plane T_i of M at $\mathbf{v}_i$
- The absolute value of the ratio between the two sorted eigenvalues, $|\mu_1| \geq |\mu_2|$ of this matrix is a good indication of a **corner response**
- By construction, this ratio is independent of the choice of the local coordinate frame, i.e., vectors $\mathbf{x}$ and $\mathbf{y}$
- $|\mu_1/\mu_2| = 10$ is used as threshold value to **eliminate** the **non-stable edge responses**

Feature descriptor (mesh-HOG)

- In association with the **mesh-DOG detector**, a local **descriptor** is defined (**mesh-HOG**) on 2D manifolds (similar to 2D HOG)
- A 2D image is in essence a 2D regular grid
- The regularity assumption does not always hold in the case of a 2D manifold, that can exhibit non regular sampling
- For this reason, the **support region** of the descriptor has to be chosen using a measure invariant to local triangulation, such as the geodesic distance
- In addition to invariance to mesh sampling, the descriptor should also exhibit **invariance** to a number of other transformations, such as **rotation** and **scale**

Feature descriptor

- The **scale invariance** is achieved by considering the gradient information at the scale of the detected interest point
- **Rotation invariance** is achieved by defining a **local coordinate system** using the **normal at the detected interest point**, the **dominant gradient** in the support region and **their cross product**
- Finally, a **two level histogram of gradient** is computed, both spatially, at a coarse level, in order to maintain a certain high-level spatial ordering, and using orientations, at a finer level
  - Trilinear interpolation is used in order to minimize the sampling effect in the histogram bins
  - The fact that the gradient vectors are 3D allows the computation of the histograms in 3D

Support region

- The descriptor for vertex $\mathbf{v}_i$ is computed within a **support region** $N(\mathbf{v}_i)$, defined using a **geodesic ball** of radius r, as in Eq.(4)
- The geodesic support region is chosen adaptively based on a global measure, such that the descriptor is robust to scale and to spatial sampling
- The value of r is chosen such that it covers a proportion α_r of the total **mesh area**, where $\alpha_r \in (0, 1)$

Support region

- By denoting with A_M the total area of the mesh M, which can be computed as the sum of all triangle areas, the radius of the circular support region is

$$r = \left[\sqrt{\frac{\alpha_r A_M}{\pi}} \right] \qquad (11)$$

  where $[x]$ denotes the integer of x and assuming that the **surface covering the ring neighborhood** can be **approximated with a circle**
- In practice, r such that $\alpha_r = 2\%$ is used

Support region

- It is assumed that we are dealing with fully reconstructed objects, thus recovering the "true" global object scale
- If this is the case, choosing the size of a support region based on Eq.(11) makes it scale and sampling invariant
- However, when dealing with partially reconstructed objects, the support-region size should be chosen based on the scale reported by the detected interest point

Local coordinate system

- As mentioned earlier, a **local coordinate system** is desirable, in order to make the descriptor **invariant to mesh rotations**
- A local coordinate system can be built using the unit vector $\mathbf{n}_i$ orthogonal to the plane T_i tangent to M at vertex $\mathbf{v}_i$, and a pair of orthonormal vectors residing in this plane
- Given an arbitrary **unit vector** $\mathbf{a}_i \in T_i$, a local coordinate system C is constructed as

$$C = \{\mathbf{a}_i, \mathbf{n}_i, \mathbf{a}_i \times \mathbf{n}_i\} \quad (12)$$

Local coordinate system

- It is therefore important to choose the unit vector $\mathbf{a}_i$ based on some ***intrinsic*** **local property** of the scalar function, and hence make the choice of the local coordinate system **invariant to mesh rotations**
- The direction corresponding to the most dominant **gradient magnitude** in the neighborhood exhibits such a desired behavior
- Therefore, the unit vector $\mathbf{a}_i$ is chosen as the **direction associated with the dominant bin** in a **polar histogram** h_a, with b_a = 36 bins

Local histogram

- The histogram is constructed by considering the **projected participating neighboring vertices** $\mathbf{v}_j \in N(\mathbf{v}_i)$ onto T_i
- The vertex contribution $c_i(\mathbf{v}_j)$ to the appropriate bin takes into account the **gradient magnitude** and the **geodesic distance** from the center vertex $\mathbf{v}_i$, **weighted by a Gaussian**

$$c_i(\mathbf{v}_j) = \left\|\nabla_M f(\mathbf{v}_j)\right\| G_\sigma(d_g(\mathbf{v}_i, \mathbf{v}_j))$$

where the standard deviation is $\sigma = \varepsilon r$, with r the support region radius and ε the spatial influence (set to 0.5)

Local coordinate system

- Therefore, bin $h_a(k, \mathbf{v}_i)$ yields

$$h_a(k, \mathbf{v}_i) = \sum_{\mathbf{v}_j \sim \mathbf{v}_i} \chi_{h_a(k,\mathbf{v}_i)}(\mathbf{v}_j) c_i(\mathbf{v}_j) \qquad (13)$$

  where $k = \{1, 2, \ldots, b_a\}$ is the bin index and $\chi_{h_a(k,\mathbf{v}_i)}(\mathbf{v}_j)$ represents the indicator function for bin selection
- In general, the indicator function is defined as

$$\chi_Z(x) = \begin{cases} 1 & \text{if } x \in bin\ Z \\ 0 & \text{otherwise} \end{cases}$$

- In order to reduce aliasing and the boundary effects of binning, **votes** $c_i(\mathbf{v}_j)$ are **interpolated trilinearly** between neighboring bins during the histogram computation
  - In practice, that means relaxing the indicator function definition

105

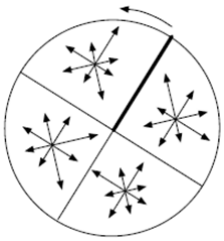
The HOG descriptor

- Instead of computing full 3D orientation histograms, the **gradient vector** is projected onto the **three planes** associated with the **local coordinate system** of Eq.(12), in order to provide a more compact representation of the descriptor
- A possible drawback of the approach described below is a decrease in the discriminability of the descriptor
- However, given the fact that local surface neighborhoods and the associated gradients estimated at the mesh vertices typically span a limited subset of the possible 3D spatial/orientation bins, the current compression scheme does not incur any practical setbacks

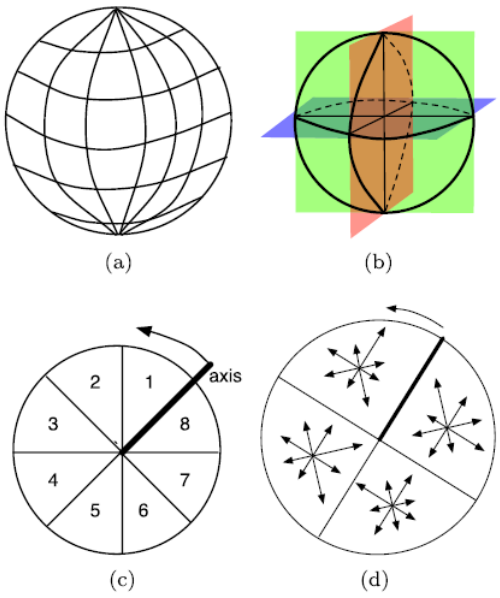
Computer Graphics and 3D 106

The HOG descriptor

- For each of the three planes, a **two-level histogram** $h_{s,o}$ is computed
  - First, the plane is divided in b_s = 4 **polar slices** h_s , starting with the origin and continuing in the direction dictated by the right hand rule, with respect to the other orthonormal vector. When projected onto the plane, the participating neighboring vertices $\mathbf{v}_j$ will fall within one of the spatial slices
  - Second, for each spatial slice, the space is divided into b_o = 8 **orientation slices** h_o. The projected gradient vectors $\nabla_M f(\mathbf{v}_j)$ of the vertices $\mathbf{v}_j \in N(\mathbf{v}_i)$ that projected onto spatial slice are used to determine the orientation slice



The HOG descriptor



(a) 3D Histogram—polar mapping used for creating histograms via binning of 3D vectors
(b) Choosing three orthogonal planes onto which to project the 3D histogram
(c) Polar coordinate system used for creating histograms via binning of 2D vectors, shown in this example with eight polar slices
(d) Example of typical spatial and orientation histograms, using **four spatial polar slices** and **eight orientation slices**

The HOG descriptor

- Similar to the histogram definition in Eq.(13), the histogram bin $h_{s,o}(e, l, \mathbf{v}_i)$ yields

$$h_{s,o}(e,l,\mathbf{v}_i) = \sum_{\mathbf{v}_j \sim \mathbf{v}_i} \chi_{h_{s,o}(e,l,\mathbf{v}_i)}(\mathbf{v}_j) c_i(\mathbf{v}_j)$$

where $e = \{1, 2, \ldots, b_s\}$ is the spatial bin index, $i = \{1, 2, \ldots, b_o\}$ is the orientation bin index and $\chi_{h_{s,o}(e,l,\mathbf{v}_i)}(\mathbf{v}_j)$ represents the indicator function for bin selection, defined as

$$\chi_{h_{s,o}(e,l,\mathbf{v}_i)}(\mathbf{v}_j) = \chi_{h_s(e,\mathbf{v}_i)}(\mathbf{v}_j) \chi_{h_o(l,\mathbf{v}_i)}(\nabla_M f(\mathbf{v}_j))$$

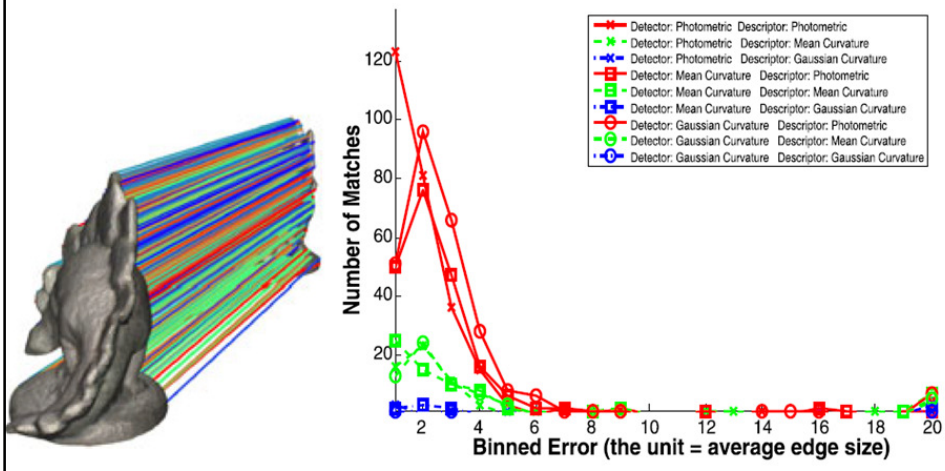
The HOG descriptor

- The final descriptor is obtained by **concatenating** the $b_s \cdot b_o$ histogram values for each of the 3 orthonormal planes
- In order to make the descriptor invariant to mesh sampling, the **concatenated histograms are normalized** using the L_2 norm
- The final descriptor will have $3 \times b_s \times b_o$ elements
- Given the previous choice of parameters, the dimensionality of the descriptor is $3 \times 4 \times 8 = 96$

The HOG descriptor

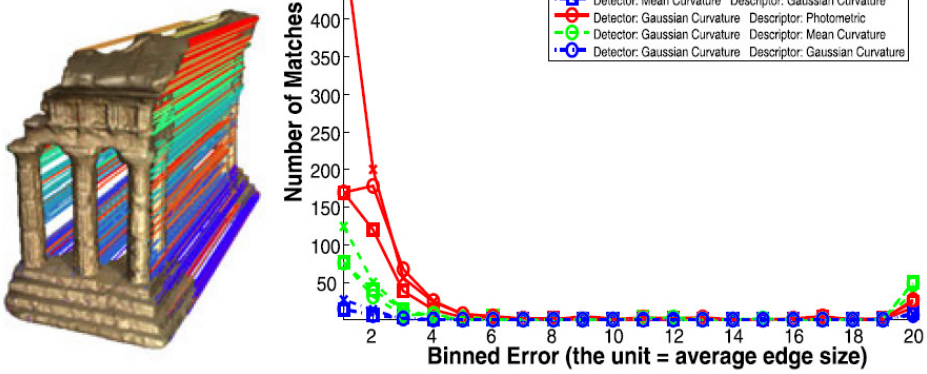
- Whenever reducing the descriptor dimensionality is a requirement, keeping only **histograms computed in the tangent plane** T_i is a possibility, thus shrinking the descriptor to 32 elements
- Also, if multiple scalar functions are available, an aggregate descriptor can be built by concatenating multiple individual descriptors

Rigid matching: Corresponding extrema



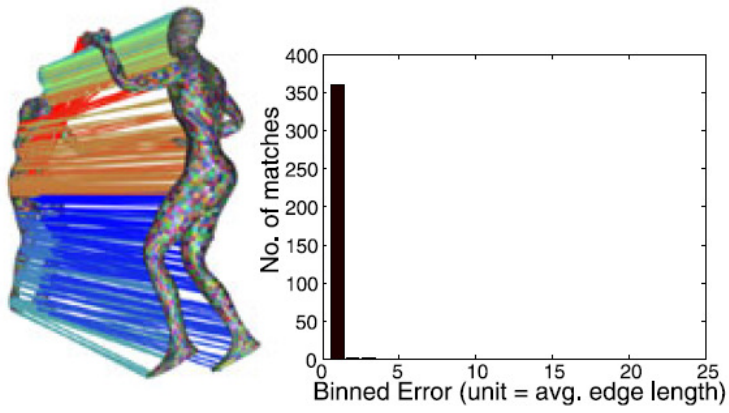
Rigid matching results—*Dino* datasets.
(**left**) Matching when using photometric information for both detection and description
(**right**) Error distribution when using different combinations of scalar functions 112

Rigid matching: Corresponding extrema



Rigid matching results—*Temple* datasets.
(**left**) Matching when using photometric information for both detection and description
(**right**) Error distribution when using different combinations of scalar functions

Non-rigid matching: Corresponding extrema



Non Rigid matching using synthetic data—*dancer-synth* dataset.
(**left**) Matches between frames 1 and 50 are visually depicted. There are 364 matches.
(**right**) In the error histogram, the histogram bins are of size equal to e_{avg}. The last bin groups all the errors greater than $20 * e_{avg}$

Non-rigid matching: Corresponding extrema

Non Rigid matching using real data—*Dance-1* sequence
(**left**) Matches between frames 525 and 526 (174 matches)
(**right**) Matches between frames 530 and 550 (27 matches)

- A comparative evaluation between 3D keypoints detectors, including mesh-DOG, is given in [14]

DETECTING SALIENT POINTS OF A SURFACE: MESH-SIFT (OPTIONAL)

Mesh-SIFT idea

- **Mesh-SIFT** [6] develops on the idea of the **Scale Invariant feature Transform** (SIFT) [7], extending to the case of 3D meshes
- It consists of four major components
  - Keypoints detection
  - Orientation assignment
  - Local feature description
  - Feature matching

Notation

- The mesh-SIFT algorithm starts from a surface mesh $M = (V, E, T)$, consisting of a collection of vertices $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_{|V|}\}$ connected by edges E and a set of simplices $T = \{t_1, \ldots, t_{|T|}\}$ covering the surface such that $T = \cup t_i, t_i \cap t_j = \mathrm{O}$ for $i \neq j$

- Let N be a neighborhood system defined on V, where $N_i = \{\mathbf{v}_j \in V | (\mathbf{v}_i, \mathbf{v}_j) \in E\}$ denotes the set of neighbors of $\mathbf{v}_i$

Keypoint detection

- The keypoint detection component identifies **salient points** on the mesh
- Similar to the 2D SIFT algorithm, a **scale space** is constructed containing smoothed versions M_s of the input mesh M, expressed as

$$M_s = \begin{cases} M & s = 0 \\ \hat{G}_{\sigma_s} \otimes M & otherwise \end{cases}$$

  where M is the original mesh and $\hat{G}_{\sigma_s}$ is the approximated **Gaussian filter** with scale σ_s (standard deviation)
- These scales σ_s vary exponentially as $\sigma_s = 2^{s/k}\sigma_0$, with k and σ_s parameters of the mesh-SIFT algorithm

Keypoint detection

- The **Gaussian filter** for meshes is approximated as subsequent convolutions of the mesh with a **binomial filter**
- This **binomial filter** moves each vertex $\mathbf{v}_i$ towards

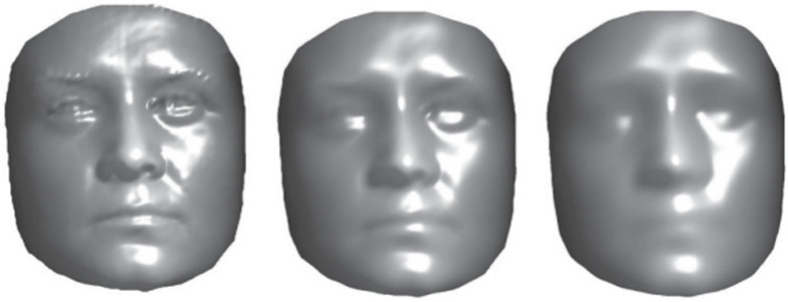
$$\mathbf{v}_i^s = \frac{1}{|N_i|}\left(\mathbf{v}_i + \sum_{j \in N_i} \mathbf{v}_j\right)$$

- Since the number of convolutions is discrete, σ_s is approximated as

$$\tilde{\sigma}_s = \bar{e} \cdot \sqrt{\frac{2 \cdot \sigma_0}{3}} \cdot 2^{s/k}$$

  to approximate as close as possible the desired exponential behavior, with $\bar{e}$ the average edge length and $s = 0, \ldots, n_{scales} + 2$

Keypoint detection



M^1, $\tilde{\sigma}_s = 1.83mm$ M^5, $\tilde{\sigma}_s = 3.66mm$ M^8, $\tilde{\sigma}_s = 6.18mm$

Smoothed meshes in the scale space

Keypoint detection

- Next, for the detection of salient points in the scale space, the **mean curvature** H is computed [8] for each vertex *i* and at each scale *s* in the scale space $H_i^s = \frac{k_{i,1}^s + k_{i,2}^s}{2}$

  with $k_{i,1}^s$ and $k_{i,2}^s$ the maximal and minimal principal curvature at vertex *i* and scale s
- Note that also other functions can be defined on the mesh, such as the **Gaussian curvature** K, the ***shape index*** S

  $S_i = \frac{2}{\pi}\tan^{-1}\left(\frac{k_{i,1} + k_{i,2}}{k_{i,1} - k_{i,2}}\right)$

  or the ***curvedness*** $C = \sqrt{2H^2 - K} = \sqrt{\frac{k_1^2 + k_2^2}{2}}$

Keypoint detection

- The **mean curvature** appeared to provide the most stable keypoints (at least for 3D face data)
- Note also that the mesh itself is smoothed and not the function on the mesh as for mesh-DOG
  - As such, mesh-SIFT better describes the shape's geometry
- It is, however, not completely intrinsic anymore (and therefore, not completely invariant for local isometric deformations)

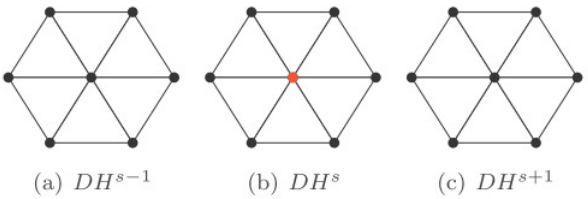
Keypoint detection

- Differences between subsequent scales are computed,

$$dH_i^s = H_i^{s+1} - H_i^s$$

and **extrema** (minima and maxima) in this scale space are selected as local feature locations

- This means that each vertex with a higher (or lower) value of dH^s_i than all of its neighbors on the same scale as well as on the upper and lower scale is selected as **keypoint**

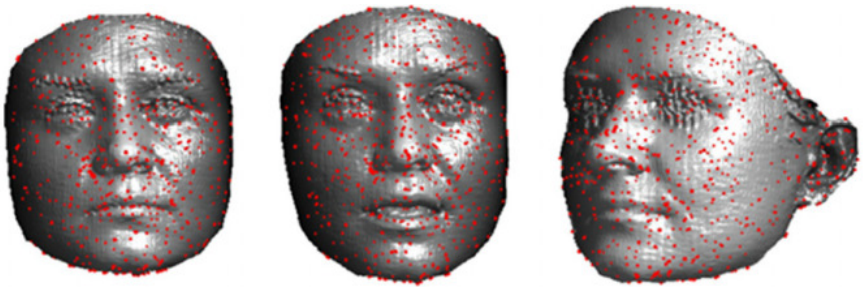


(a) DH^{s-1} (b) DH^s (c) DH^{s+1}

The neighborhood of a vertex in scale space

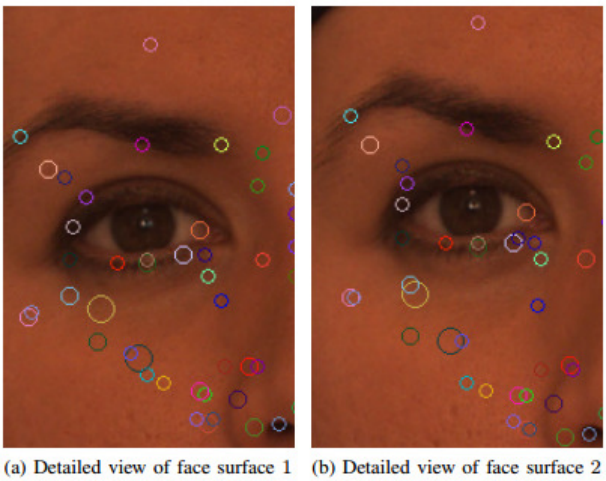
Keypoint detection

- Finally, the **scale** σ_s at which this extremum is obtained is assigned to each keypoint



Detected extrema of a neutral face, a surprised face and a face 30° rotated.
The number of detected features depends on the chosen parameters (k, σ_0 , n_{scales})

Keypoint detection



(a) Detailed view of face surface 1 (b) Detailed view of face surface 2

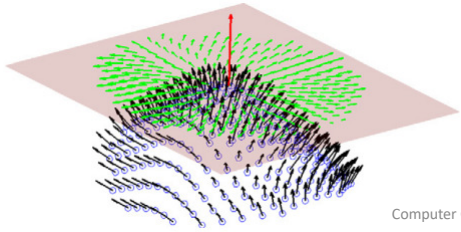
Detailed view of two faces with neutral expression. Corresponding scale space extrema are shown in the same color

Orientation assignment

- In order to obtain an **orientation-invariant descriptor**, each keypoint is assigned a **canonical orientation**
- It allows, together with the **normal** to the surface at the keypoint, to construct a local reference frame in which the vertices of the neighborhood can be expressed independent of the (facial) pose
- By expressing the neighborhood size as a function of the scale σ_s, a **scale invariant** descriptor is ensured as well

Orientation assignment

- Only **vertices** within **a spherical region** with radius $9\sigma_s$ around each **keypoint** are considered
  - First, for each vertex within this (neighborhood) region, the normal vector is computed [8] and the geodesic distance to the respective keypoint is determined based on the **fast marching** algorithm for triangulated domains [9]
  - Next, all calculated **normal vectors** in the neighborhood are projected onto the **tangent plane** to the mesh containing the keypoint (see Figure below)



Orientation assignment

- These projected normal vectors are gathered in a **weighted histogram** comprising 360 bins (thus covering 360° with a bin width of 1°)
- Each histogram entry is **Gaussian weighted** with its **geodesic distance** to the **keypoint**, with a bandwidth proportional to the assigned scale ($\sigma = 4.5\sigma_s$)
- The resulting **histogram is smoothed** by convolving it three times with a **Gaussian filter** (17 bins, $\sigma = 17$) for a more robust localization of the canonical orientation
- Finally, the **highest peak in the histogram** and every peak above 80% of this highest peak value are selected as **canonical orientations**

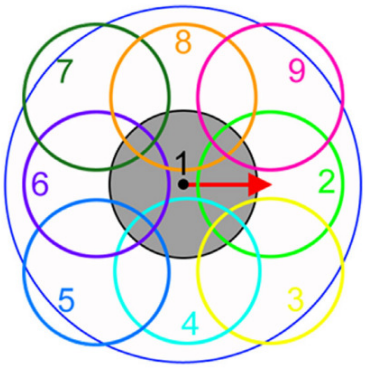
Orientation assignment

- By **fitting a quadratic function** to the **histogram** using the neighboring bins of a peak, the canonical orientation is calculated to sub-bin precision
- If more than one canonical orientation exists for a keypoint, this results in **multiple keypoints**, each assigned one of the canonical orientations
- Intuitively, the **canonical orientation** can be seen as the direction in which the **surface bends** the most

Feature description

- The feature descriptor summarizes the local neighborhood around a keypoint
- It provides for each **keypoint** (with assigned scale and canonical orientation) a **feature vector** consisting of concatenated histograms
- Each of these histograms is calculated over one of the nine small circular regions, defined with respect to the canonical orientation as shown in Figure

Location and order of the regions w.r.t. the **canonical orientation**, used for the construction of the **feature vector**



Feature description

- Regions have a geodesic radius of $3.75\sigma_s$ and their centers are located at a geodesic distance of $4.5\sigma_s$ (regions 2, 4, 6 and 8) or $4.5\sqrt{2}\sigma_s$ (regions 3, 5, 7 and 9) respectively, to the keypoint
- In each region, two histograms, $\mathbf{P}_S$ and $\mathbf{P}_\theta$, with eight bins each are computed
- The first histogram contains the shape index, which is expressed for vertex *i* as

$$S_i = \frac{2}{\pi}\tan^{-1}\left(\frac{k_{i,1}+k_{i,2}}{k_{i,1}-k_{i,2}}\right)$$

with $k_{i,1}$ the **maximum** and $k_{i,2}$ the **minimum** curvature

Feature description

- The second histogram contains the **slant angles**, which are defined as the **angle** between **every projected normal** and the **canonical orientation**
- First, each entry (the shape index or slant angle of a vertex) for both histograms is Gaussian weighted with the geodesic distance to the keypoint (large circle in previous Figure, $\sigma = 4.5\sigma_s$) and with the geodesic distance to the centre of the region (small circles in previous Figure, $\sigma = 4.5\sigma_s$)

Feature description

- Moreover, the entries of the **shape index histogram** are weighted with the **curvedness** $C = \sqrt{2H^2 - K}$ and the entries of the **slant angle histogram** are weighted with the **tilted angle**, i.e., the angle between the normal in the considered vertex and the normal in the keypoint
- Next, every histogram is normalized and clipped to $1/\sqrt{8}$ reducing the influence of large histogram values
- Finally, the histograms are concatenated in a single feature vector $\mathbf{f}_i = [\mathbf{P}_{S,1}\ \mathbf{P}_{\theta,1} \ldots \mathbf{P}_{S,9}\ \mathbf{P}_{\theta,9}]^T$
- A feature descriptor is computed for each keypoint, resulting in a set of feature vectors per (face) surface $F = \{\mathbf{f}_1, \mathbf{f}_2, \ldots \mathbf{f}_n\}$

Feature matching

- In order to find corresponding features between two surfaces, the sets of feature vectors of both surfaces are compared using the **angle** as similarity measure
- The angle is defined as

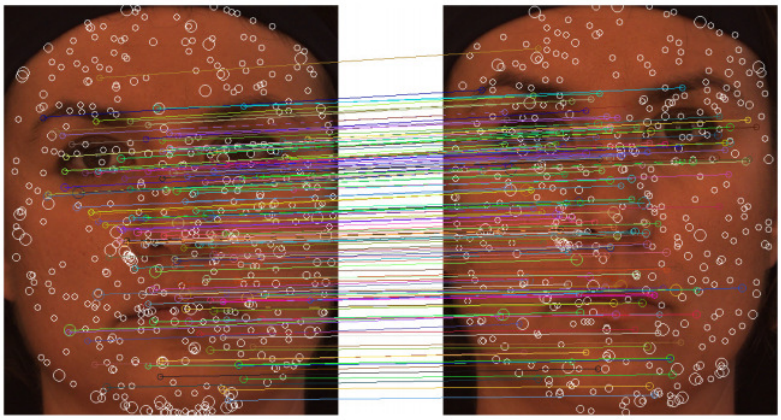
$$\alpha = \cos^{-1}\left(\frac{\langle \mathbf{f}_i, \mathbf{f}_j \rangle}{\| \mathbf{f}_i \| \| \mathbf{f}_j \|}\right)$$

- For each feature, the angles for all candidates are then ranked in ascending order
- If the ratio between the first and the second is smaller than r_{thr} , a match is accepted; other matches are rejected

Feature matching

- This is done for both surfaces in order to obtain a symmetric feature matching (independent of the choice of the order of both surfaces)
- As a consequence, a feature in one surface can, in theory, be matched with multiple features in the other surface
- Matches are mostly found between two face surfaces of the same person, allowing the algorithm to be used for 3D face recognition
- The **number of matches** is simply used as **similarity criterion**
- To allow a very fast matching between surfaces, no spatial consistency between features is considered

Applications: 3D face recognition



Corresponding scale space extrema for two face surfaces with a neutral expression (both from the same person)

References

[1] J. Koenderink and A. van Doorn. “Surface shape and curvature scales,” *Image and Vision Computing*, vol.10, no.8, pp.557–565, 1992

[2] R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin. «Shape Distributions,» ACM Transactions on Graphics (TOG), vol.21, no.4, pp.807-832 October 2002

[3] T. Ojala, M. Pietikainen, and D. Harwood, “A comparative study of texture measures with classification based on featured distribution,” *Pattern Recognition*, vol. 29, no. 1, pp. 51–59, Jan. 1996

[4] T. Ojala, M. Pietikainen, and T. Maenpaa, “Multiresolution gray-scale and rotation invariant texture classification with local binary patterns,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 971–987, July 2002

[5] N. Werghi, S. Berretti, A. Del Bimbo. "The mesh-LBP: a Framework for Extracting Local Binary Patterns from Discrete Manifolds," *IEEE Trans. on Image Processing*, vol.24, no.1, pp.220-235, January 2015

References

[6] D. Smeets, J. Keustermans, D. Vandermeulen, P. Suetens. "meshSIFT: Local surface features for 3D face recognition under expression variations and partial data," *Computer Vision and Image Understanding,* vol. 117,158–169, 2013

[7] D.G. Lowe. "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision,* vol.60, n.2, 91–110, 2004

[8] G. Peyre, Toolbox graph. MATLAB Central File Exchange Select, 2009

[9] G. Peyre, Toolbox fast marching. MATLAB Central File Exchange Select, 2009

[10] Andrei Zaharescu, Edmond Boyer, Kiran Varanasi, Radu Horaud, "Surface Feature Detection and Description with Applications to Mesh Matching,"*IEEE Int. Conf. on Computer Vision and Pattern Recognition* (CVPR'09), Miami, Florida, June 2009

References

[11] A. Zaharescu, E. Boyer, and R. Horaud. "Keypoints and Local Descriptors of Scalar Functions on 2D Manifolds," *International Journal of Computer Vision*, vol.100, no.1, pp.78–98, 2012

[12] M. Meyer, M. Desbrun, P. Schröder, and A.H. Barr. "Discrete differential geometry operators for triangulated 2-dimensional manifolds," In *Proceedings of VisMath,* 2002

[13] C.S. Dong, and G.Z. Wang. "Curvatures estimation on triangular mesh," *Journal of Zhejiang University SCIENCE*, vol.6A, no.1, pp.128–136, 2005

[14] Federico Tombari, Samuele Salti, and Luigi Di Stefano, "Performance Evaluation of 3D Keypoint Detectors," *International Journal of Computer Vision*, vol.102, pp.198–220, 2013

Notes on 3D printing

Lecture 16-B

What is 3D printing?

- 3D printing or **additive manufacturing** is a process of making 3D solid objects from a digital file
- The creation of a 3D printed object is achieved using **additive processes**, where an object is created by laying down successive layers of material until the entire object is created
  - Each of these layers can be seen as a thinly sliced horizontal cross-section of the eventual object
- It all starts with making a virtual design of the object you want to create
  - This virtual design is made in a CAD (Computer Aided Design) file using a 3D modeling program (for the creation of a totally new object) or with the use of a 3D scanner (to copy an existing object)

3D printing file formats

- STL (STereoLithography) is a file format native to the stereolithography CAD software created by **3D Systems**
- This file format is supported by many other software packages; it is widely used for rapid prototyping, 3D printing and computer-aided manufacturing
- STL files describe only the surface geometry of a 3D object without any representation of color, texture or other common CAD model attributes
- The STL format exists in both ASCII and binary representations
  - Binary files are more common, since they are more compact

3D printing file formats

- **Additive Manufacturing File Format** (AMF) is an open standard for describing objects for additive manufacturing processes such as 3D printing
- The official ISO/ASTM 52915:2013 standard is an XML-based format designed to allow any CAD software to describe the shape and composition of any 3D object to be fabricated on any 3D printer
- Unlike STL format, AMF has native support for color, materials, lattices, and constellations

How does 3D printing work?

- To prepare a digital file for 3D printing, the 3D modeling software “slices” the final model into hundreds or thousands of **horizontal layers**
- When the sliced file is uploaded in a 3D printer, the object can be created **layer-by-layer**
- The 3D printer reads every slice (or 2D image) and creates the object, blending each layer with hardly any visible sign of the layers, with as a result the 3D object

Processes and technologies

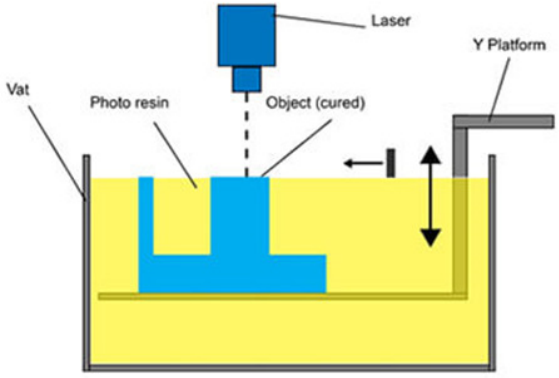
- There are several ways to print and all those available are **additive**, differing mainly in the way layers are built to create the final object
- Some methods use melting or softening material to produce the layers
  - **Selective laser sintering** (SLS), and **fused deposition modeling** (FDM) are the most common technologies using this way of 3D printing (respectively, categories 5 and 4 in the following)
- Another method is when we talk about curing a photo-reactive resin with a UV laser or another similar power source one layer at a time
  - The most common technology using this method is called **stereolithography** (SLA) (category 1 in the following)

Processes and technologies

- Since 2010, the **American Society for Testing and Materials** (ASTM) group "**ASTM F42 – Additive Manufacturing**", developed a set of standards that classify the Additive Manufacturing processes into **7 categories** according to **Standard Terminology for Additive Manufacturing Technologies**. These seven processes are
  1. **Vat Photopolymerisation**
  2. **Material Jetting**
  3. **Binder Jetting**
  4. **Material Extrusion**
  5. **Powder Bed Fusion**
  6. **Sheet Lamination**
  7. **Directed Energy Deposition**

1. Vat Photopolymerisation

- A 3D printer based on the **Vat Photopolymerisation** method has a container filled with photopolymer resin which is then hardened with UV light source



1. Vat Photopolymerisation

- The most commonly used technology in this process is **Stereolithography** (**SLA**)
- This technology employs a vat of liquid ultraviolet curable photopolymer resin and an ultraviolet laser to build the object's layers one at a time
- For each layer, the laser beam traces a cross-section of the part pattern on the surface of the liquid resin
- Exposure to the ultraviolet laser light cures and solidifies the pattern traced on the resin and joins it to the layer below

1. Vat Photopolymerisation

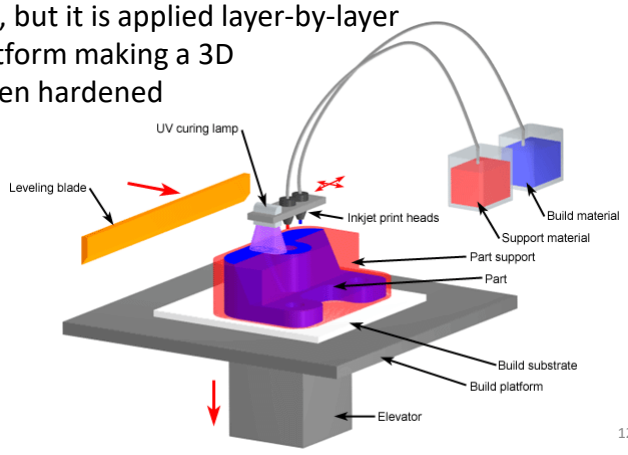
- After the pattern has been traced, the SLA's elevator platform descends by a distance equal to the thickness of a single layer, typically 0.05 mm to 0.15 mm
- Then, a resin-filled blade sweeps across the cross section of the part, re-coating it with fresh material
- On this new liquid surface, the subsequent layer pattern is traced, joining the previous layer
- The complete 3D object is formed by this process

1. Vat Photopolymerisation

- Stereolithography requires the use of supporting structures, which serve to attach the part to the elevator platform and to hold the object because it floats in the basin filled with liquid resin
  - These are removed manually after the object is finished
- This technique was invented in 1986 by *Charles Hull*, who also at that time founded the **3D Systems** company
- Other technologies using Vat Photopolymerisation are the new ultrafast **Continuous Liquid Interface Production** (CLIP), and marginally used older **Film Transfer Imaging** and **Solid Ground Curing**

2. Material Jetting

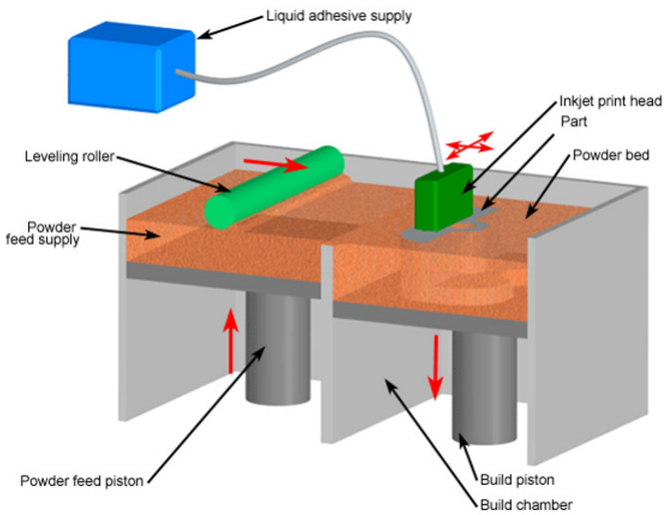
- In this process, material is applied in droplets through a small diameter nozzle, similar to the way a common inkjet paper printer works, but it is applied layer-by-layer to a build platform making a 3D object and then hardened by UV light



3. Binder Jetting

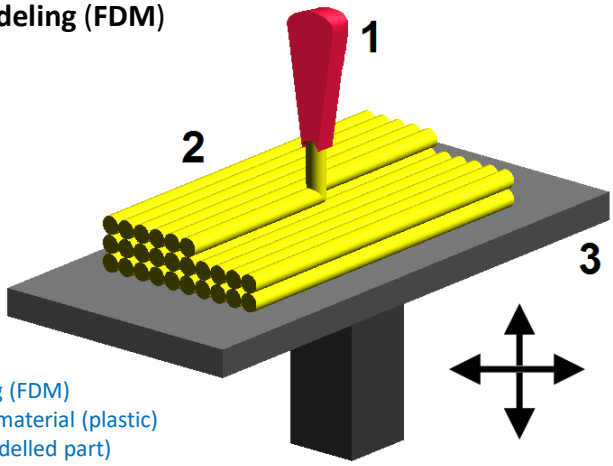
- With binder jetting **two materials** are used: powder base material and a liquid binder
- In the build chamber, powder is spread in equal layers and **binder** is applied through jet nozzles that "**glue**" the **powder particles** in the shape of a programmed 3D object
- The finished object is "glued together" by binder, and remains in the container with the powder base material
- After the print is finished, the remaining powder is cleaned off and used for 3D printing the next object
- This technology was first developed at the *Massachusetts Institute of Technology* in 1993, and in 1995 **Z Corporation** obtained an exclusive license

3. Binder Jetting



4. Material Extrusion

- The most commonly used technology in this process is **Fused Deposition Modeling** (**FDM**)



Fused deposition modelling (FDM)
1 – nozzle ejecting molten material (plastic)
2 – deposited material (modelled part)
3 – controlled movable table

4. Material Extrusion

- The FDM technology works using a **plastic filament** or **metal wire**, which is unwound from a coil and supplying material to an extrusion nozzle. This latter can turn the flow on and off
- The **nozzle is heated** to melt the material and can be moved in both horizontal and vertical directions by a numerically controlled mechanism, directly controlled by a computer-aided manufacturing (CAM) software package
- The object is produced by **extruding melted material** to form layers as the **material hardens** immediately after extrusion from the nozzle

4. Material Extrusion

- This technology is most widely used with **two plastic filament** material types
  - ABS (Acrylonitrile Butadiene Styrene)
  - PLA (Polylactic acid)

  but many other materials are available ranging in properties from wood filed, conductive, flexible, etc.

- The software that comes with this technology automatically generates support structures if required
- The machine dispenses two materials, one for the model and one for a disposable support structure

4. Material Extrusion

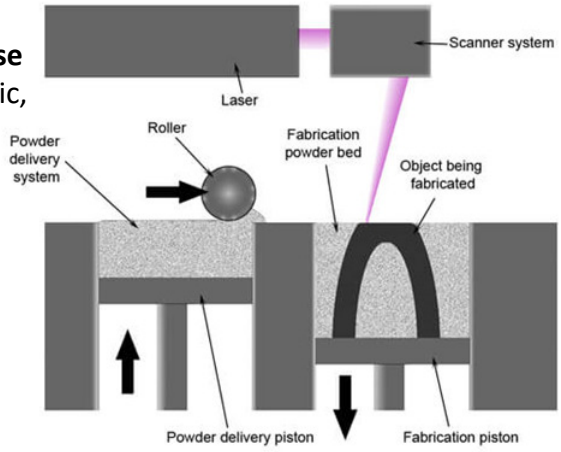
- FDM was invented by *Scott Crump* in the late 80's
  - After patenting this technology, he started the company **Stratasys** in 1988
- The term fused deposition modeling and its abbreviation to FDM are trademarked by Stratasys Inc
- The exactly equivalent term, **fused filament fabrication** (FFF), was coined by the members of the **RepRap** project to give a phrase that would be legally unconstrained in its use

Note

- A 3D printer using this technology is available at MICC

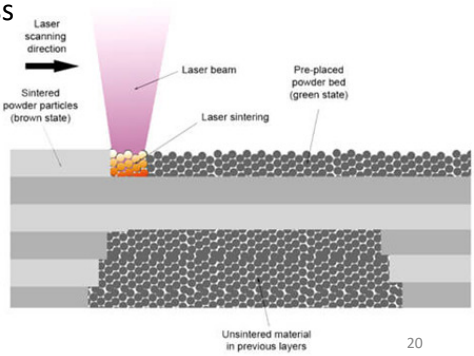
5. Powder Bed Fusion

- The most commonly used technology in this process is **Selective Laser Sintering (SLS)**
- This technology uses a **high power laser** to **fuse small particles** of plastic, metal, ceramic or glass powders into a mass that has the desired three dimensional shape



5. Powder Bed Fusion

- The laser selectively fuses the powdered material by scanning the cross-sections (or layers) generated by the 3D modeling program on the surface of a powder bed
- After each cross-section is scanned, the powder bed is lowered by one layer thickness
- Then, a new layer of material is applied on top, and the process is repeated until the object is completed



20

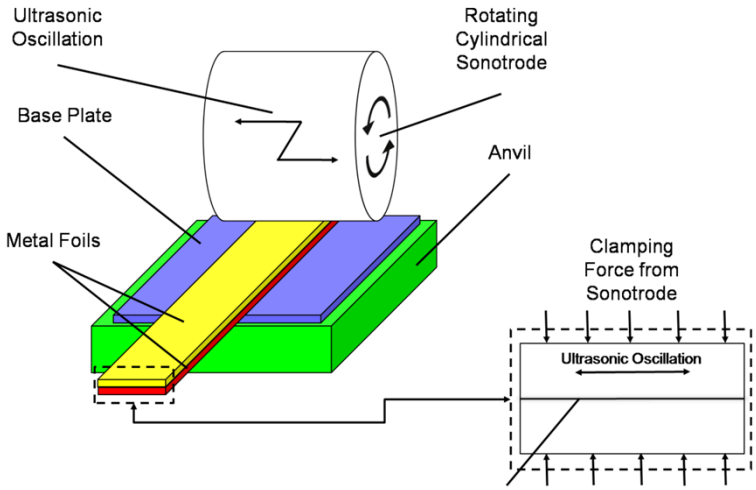
5. Powder Bed Fusion

- All untouched powder remains as it is and becomes a support structure for the object
- Therefore there is no need for any support structure, which is an advantage of SLS over SLA
- All unused powder can be used for the next print
- SLS was developed and patented by *Carl Deckard* at the University of Texas in the mid-1980s, under sponsorship of DARPA

6. Sheet Lamination

- **Sheet lamination** involves **material in sheets**, which is bound together with **external force**
- Sheets can be metal, paper or a form of polymer
- Metal sheets are welded together by **ultrasonic welding** in layers and then milled into a proper shape using *computer numerical control* (CNC) machines
- Paper sheets can be used also, but they are glued by adhesive glue and cut in shape by precise blades
- A leading company in this field is **Mcor Technologies**

6. Sheet Lamination

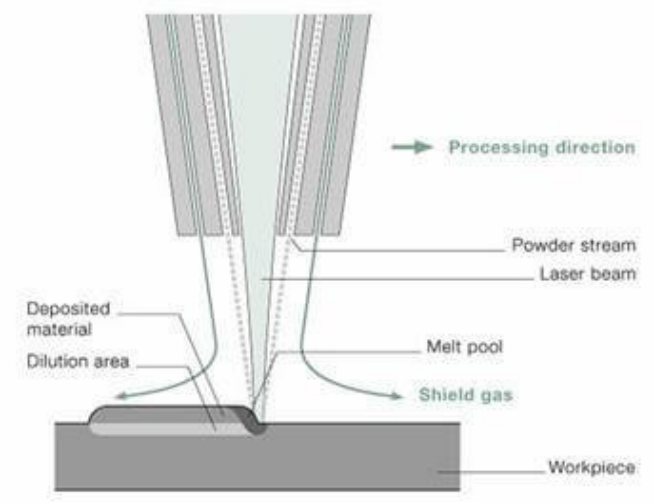


Simplified model of ultrasonic sheet metal 3D printing

7. Directed Energy Deposition

- This process is mostly used in the **high-tech metal industry** and in rapid **manufacturing applications**
- The 3D printing apparatus is usually attached to a multi-axis **robotic arm** and consists of a nozzle that deposits metal powder or wire on a surface and an energy source (laser, electron beam or plasma arc) that melts it, forming a solid object
- **Sciaky** is a major tech company in this area

7. Directed Energy Deposition



Example applications of 3D printing

- Applications include
  - *rapid prototyping*,
  - *architectural scale models* & *maquettes*,
  - *healthcare* (3D printed prosthetics and 3D printing with human tissue)
  - *entertainment* (e.g., film props)
- Other examples of 3D printing would include
  - *reconstructing fossils* in paleontology,
  - *replicating* ancient artifacts in archaeology,
  - *reconstructing* bones and body parts in forensic pathology
  - *reconstructing* heavily damaged evidence acquired from crime scene investigations