成均館大學校
SUNG KYUN KWAN UNIVERSITY

Geometric Modeling

Computer Graphics

Instructor: Sungkil Lee

Today

- **Fundamental elements of geometry**
  - Points, scalars, and vectors
- **Vector, Euclidean, and affine spaces**
- **Additional elements of geometry**
- **Geometric modeling**

Prerequisites: Vector Spaces

Vector Spaces

- **Formal definition of a vector space**
  - A vector space over a field F is a set V together with addition and multiplication that satisfy the eight axioms.
  - Elements of V and F are called vectors and scalars.

Axiom	Meaning
Associativity of addition	$\mathbf{u} + (\mathbf{v} + \mathbf{w}) = (\mathbf{u} + \mathbf{v}) + \mathbf{w}$
Commutativity of addition	$\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}$
Identity element of addition	There exists an element $\mathbf{0} \in V$ such that $\mathbf{v} + \mathbf{0} = \mathbf{v}$ for all $\mathbf{v} \in V$.
Inverse elements of addition	For every $\mathbf{v} \in V$, there exists an element $-\mathbf{v} \in V$ such that $\mathbf{v} + (-\mathbf{v}) = \mathbf{0}$.
Distributivity of scalar multiplication with respect to vector addition	$a(\mathbf{u} + \mathbf{v}) = a\mathbf{u} + a\mathbf{v}$
Distributivity of scalar multiplication with respect to field addition	$(a + b)\mathbf{v} = a\mathbf{v} + b\mathbf{v}$
Compatibility of scalar multiplication with field multiplication	$a(b\mathbf{v}) = (ab)\mathbf{v}$
Identity element of scalar multiplication	$1\mathbf{v} = \mathbf{v}$, where 1 denotes the multiplicative identity in F.

Vector Spaces

- **More simply:**
  - Vectors can be added, and such a sum is also a vector.
  - There is a zero vector and an inverse on vector addition.
  - Vectors can be multiplied by a scalar.
  - Identity exists for scalar multiplication (i.e., 1).

More on Algebra

- **Mathematical structures related to the concept of a field can be tracked as follows:**

- A *field* is a *ring* whose nonzero elements form a *abelian group* under multiplication.
- A *ring* is an *abelian group* under addition and a *semigroup* under multiplication; addition is commutative, addition and multiplication are associative, multiplication distributes over addition, each element in the set has an additive inverse, and there exists an additive identity.
- An *abelian group* (*commutative group*) is a group in which commutativity ($a \cdot b = b \cdot a$) is satisfied.
- A *semigroup* is a set A in which $a \cdot b$ satisfies associativity for any two elements a and b and opeator $\cdot$.
- A *group* is a set A in which $a \cdot b$ satisfies closure, associativity, identity element, and inverse element for any two elements a and b and opeator $\cdot$.

Geometric Elements

Geometry and Fundamental Elements

- **Geometry:**
  - The study of the relationships among objects in an n-dimensional space
- **In CG, we work with sets of geometric objects, such as points, lines, triangles, and quads.**
  - Such objects exist in a 3D space.
  - We can define them and their relationships using a limited set of primitives.
- **Three fundamental types of elements:**
  - Points, scalars, and vectors

Fundamental Elements (1): Points

- **Point: a location in space**
  - a mathematical point has neither a size nor a shape.

- **Points are useful in specifying geometric objects but are insufficient by themselves**
  - We need real numbers to specify quantities such as the distance between two points
  - Such real numbers are examples of *scalars*.

Fundamental Elements (2): Scalars

- **Scalars:**
  - Objects that obey a set of rules that abstractions of the operations of ordinary arithmetic.
  - Addition and multiplication are well defined and obey fundamental axioms (associativity, commutativity, inverse, and identity).

- **Examples of scalars:**

  - real numbers
  - complex numbers

- **Scalars alone have no geometric properties**

Fundamental Elements (3): Vectors

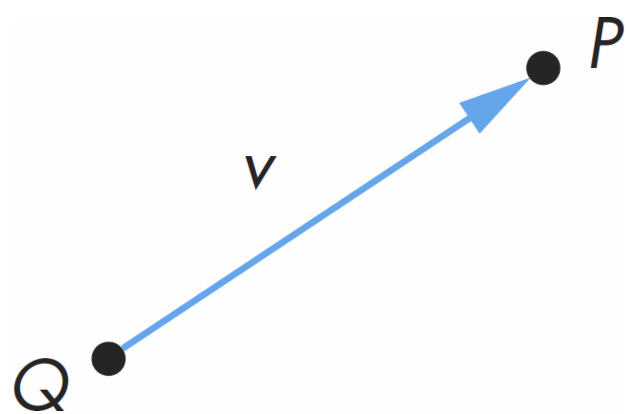
- **A physical definition of vectors:**
  - A quantity with direction and magnitude.
  - Vectors do not have a fixed location in space.

- **Examples:**
  - Force, velocity, and *directed line segments*
  - ***Directed*** *line segments*, connecting two points, will be often used synonymously to the term ***vector***.

Operations on Vectors and Points

- **Vectors are insufficient for geometry**
  - We need to represent a location in space.
  - Points necessary

- **Operations allowed between points and vectors**
  - $v = P - Q$: **point-point subtraction** yields a vector
  - $P = Q + v$: equivalent to **point-vector addition**



Computer Science View on Geom. Elements

- **We may need to define abstract data types for points, scalars, and vectors independently.**
  - The operations allowed between elements can be exactly implemented with operator overloading (in C++).
  - We can overload only allowed operators among them, and do not overload others (e.g., do not define point-point addition).

- **Notes on GLSL: confusion with vec2, vec3, vec4.**
  - Unfortunately, this choice of names by GLSL can cause some confusion.
  - They are actually not geometric types but rather storage types.
  - Hence, we can use them to store a point, a vector, or a color.

Extensions of Vector Spaces

Euclidean Space

- **Euclidean space**
  - Vector space + ***a measure of distance (i.e., Euclidean distance)***
  - Euclidean distance allows us to define size or distance as the length of a line segment.
  - When we also have the notion of point (i.e., ***affine space***), a Euclidean distance between two points can be defined as (in 3D):

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Affine Space

- **Affine space**
  - Vector space (scalars and vectors only) + points
  - Operations
    - Vector-vector addition
    - Scalar-vector multiplication
    - Scalar-scalar operations
    - **Vector-point addition (newly defined in affine space)**
  - New points are defined by vector-point addition
    - Alternatively, we can say there is point-point subtraction (equivalent to vector-point addition).
  - Note that there are no operations between points or scalars.

Representations

- **In these abstract spaces (vector space, Euclidean space, and affine space),**
  - Objects can be defined independently of any particular representations.
  - Representation (the lecture covered later) provides the tie between the abstract objects and their implementation (in real spaces).
  - Conversion between representations leads us to geometric transformations.

Additional Elements of Geometry

Lines

- **The sum of a point and a vector (or the subtraction of two points) leads to the notion of a line in an affine space.**
  - Consider all points of the parametric form

$$P(\alpha) = P_0 + \alpha \boldsymbol{d}$$

  - Here, a line can be defined as a set of all points that pass through P_0 in the direction of the vector $\boldsymbol{d}$

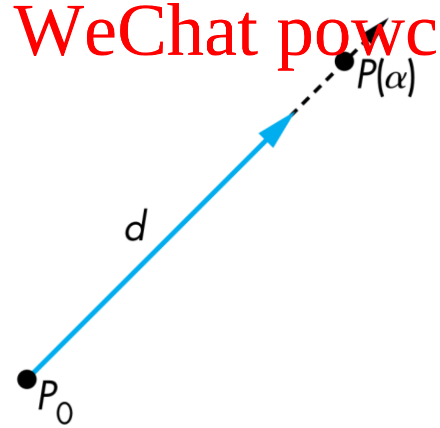


FIGURE 3.10 Line in an affine space.

Rays and Line Segments

$$P(\alpha) = Q + \alpha \boldsymbol{d} = Q + \alpha(R - Q)$$

- **If we restrict α to semi-positive values ($\alpha \geq 0$), this defines a ray emanating from Q.**
- **If we restrict α to $[0,1]$, this defines a line segment between Q and R.**

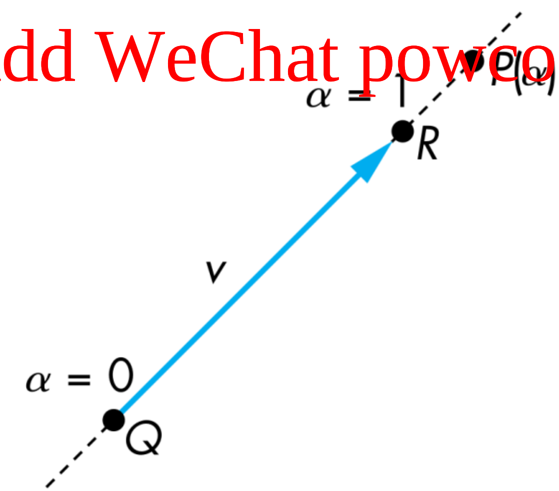


FIGURE 3.11 Affine addition.

Affine Sum

- **In an affine space, the addition of two arbitrary points and multiplication of a point by a scalar are not defined.**
  - However, we have a limited form of an operation that has certain elements of the two operations, *affine addition*.

Affine Sum

- **Affine addition:**

$$P = Q + \alpha \boldsymbol{v} = Q + \alpha(R - Q) = (1 - \alpha)Q + \alpha R$$

  - This operation looks like the addition of two points and leads to the equivalent form.

$$P = \alpha_1 Q + \alpha_2 R, \quad \text{where} \quad \alpha_1 + \alpha_2 = 1$$

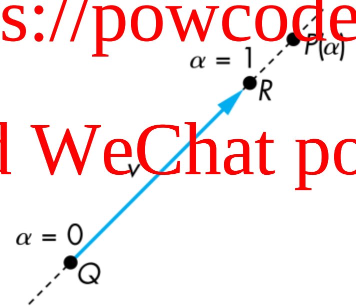


FIGURE 3.11 Affine addition.

  - Then, this defines the two operations, not allowed in an affine space:
    - (1) addition of two points and (2) multiplication of a point by a scalar
    - yet only with the limited condition (the sum of scalars=0).

Affine Sum

- **Affine sum:**
  - By extending such a point-vector addition to include n points, we have the following sum:

$$P = \alpha_1 P_1 + \alpha_2 P_2 + \cdots + \alpha_n P_n, \text{ where } \alpha_1 + \alpha_2 + \cdots + \alpha_n = 1$$

  - We call this kind of sum the affine sum.

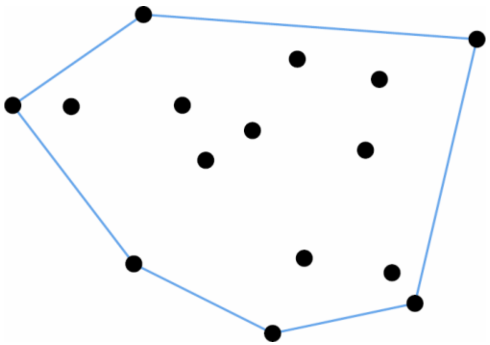
  - By this way, we can define the addition of points as well as the multiplication of points by scalars.

Convex Hull

- **Given a set of points, $\{\mathrm{P}_i\}$, one more constraint, $\alpha_i \geq 0$, defines its *convex hull*, H, as:**

$$H = \left\{ \sum_i \alpha_i \mathrm{P}_i \,\middle|\, \sum_i \alpha_i = 1, \alpha_i \geq 0 \right\}$$

  - The convex hull is the smallest convex object containing $\{\mathrm{P}_1, \mathrm{P}_2, \ldots, \mathrm{P}_n\}$.
  - **Convex object**: for any two points in the object, all points on the line segment between these points are also in the object.



Triangles: Barycentric Coordinates

- **Also, we are able to write the plane in terms of affine sum as:**

$$T(\alpha, \beta, \gamma) = \alpha P + \beta Q + \gamma R, \quad \text{where } \alpha + \beta + \gamma = 1.$$

- **When $\alpha, \beta, \gamma > 0$, this represents a triangle and its internal points.**

  - Hence, triangles are convex by default.

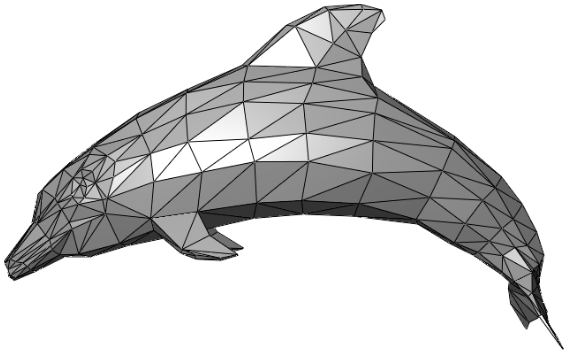
- **This representation of a point is called the barycentric coordinate representations.**
  - c.f., Barycenter: the center of mass

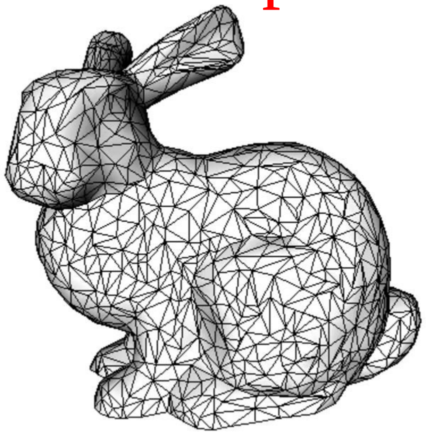
Geometric Modeling

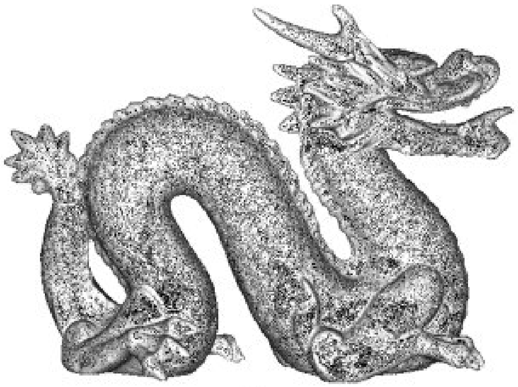
Models

- **Models:**
  - Mathematical abstraction of the real world or virtual worlds.

- **Geometric Models:**
  - In CG, we model our worlds with *geometric objects*.
  - Building blocks: a set of simple 3D primitives (Point, lines, triangles, ...)
  - ***Triangular meshes*** are common, which comprises a set of triangles connected by their common edges or corners.

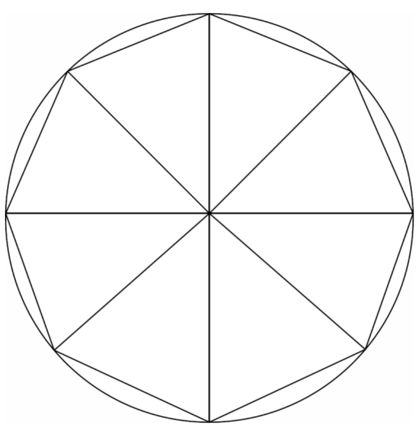


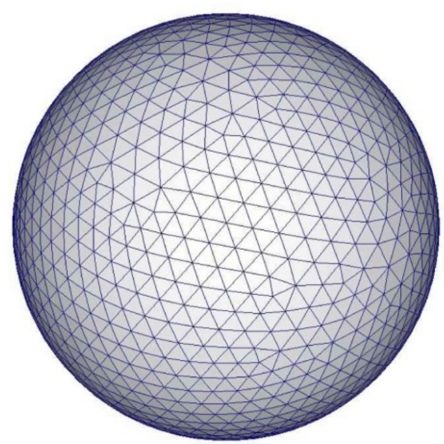




3D Primitives

- **3D objects that fit well with graphics HW and SW:**
  - described by their 2D surfaces and can be thought of as being hollow.
    - c.f., objects with 3D surfaces are called the *volumetric* objects (e.g., CT).
  - can be specified through a set of vertices.
  - either are composed of or can be approximated by flat, convex polygons.
    - e.g., a circle/sphere approximated by flat triangles.





3D Primitives

- **Why we set these conditions?**
  - Modern graphics systems are optimized for rendering **triangles** or **meshes of triangles** (e.g., more than 100 M triangles / sec.).
    - Points and lines are also supported well.
  - Vertices can be processed with the pipeline architecture, independently.

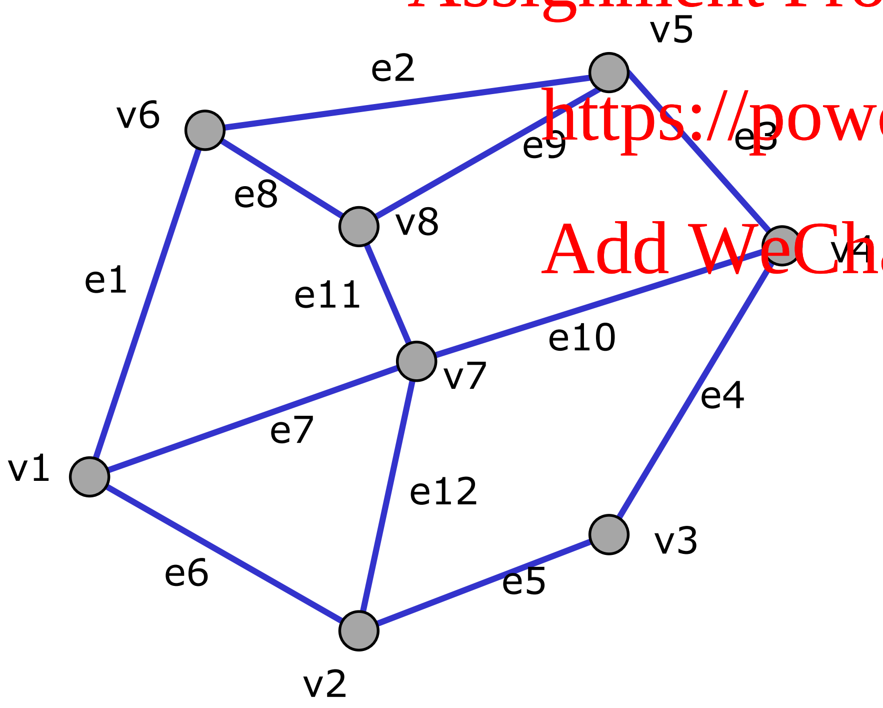
- **Why are triangles fundamental primitives?**

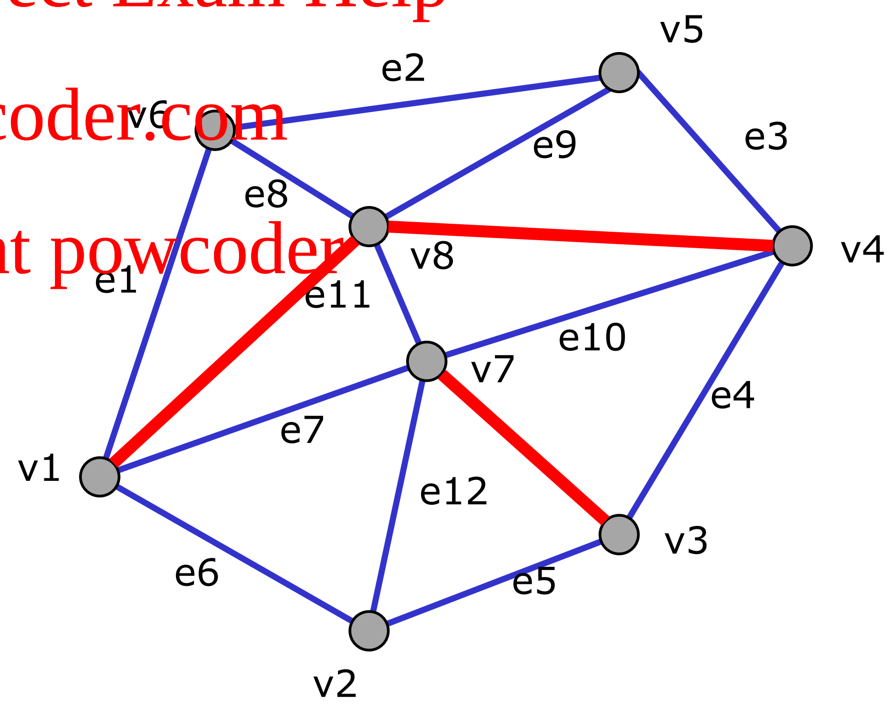
  - *The triangles are always flat*.

  - General polygons might not lie in the same plane, and then, there is no simple way to define interior of the object.
  - Also, general polygons can be decomposed into a set of triangles:
    - then, we can apply the same pipeline on the triangles.

Triangular Mesh Representation

- **Consider a mesh as a graph**
  - There are 8 vertices and 12 edges
  - 5 interior polygons and 6 interior (shared) edges
  - Decompose non-triangular polygons to triangles





Triangular Mesh Representation

- **A simple list-based representation**
  - Define each polygon by the geometric locations of its vertices, leading to OpenGL code such as.

```
...
vertex[i+0].pos = vec3(x1, y1, z1);
vertex[i+1].pos = vec3(x2, y2, z2);
vertex[i+2].pos = vec3(x7, y7, z7);
i+=3;
vertex[i+0].pos = vec3(x2, y2, z2);
vertex[i+1].pos = vec3(x3, y3, z3);
vertex[i+2].pos = vec3(x7, y7, z7);
i+=3;
...
```


  - A simple list-based representation is often inefficient and unstructured.
  - When a vertex moves to a new location, we must search and replace it for all the occurrences.

Geometry vs. Topology

- **Generally, it is a good idea to look for data structures that separate the geometry from the topology**
  - *Geometry*: locations of the vertices
  - *Topology*: structural organization of the vertices and edges
    - Connectedness is preserved under continuous deformation
    - Topology holds even if geometry changes

The cup and torus share the same topology.

Vertex Lists

- **Geometries are put into an array**
  - Use ***indices*** from the vertices into this array.
- **Topology maintains a triangle list regardless of geometry.**