Polygon Triangulation

O’Rourke, Chapter 1
Announcements

• Assignment 1 has been posted
Outline

• Polygon Area
• Implementation
Notation

Given a vector \( \vec{v} \in \mathbb{R}^2 \), we set \( \vec{v} \perp \) to be the clockwise rotation of \( \vec{v} \) by 90° degrees.

If \( \vec{v} = (x, y) \) then we have:
\[
\vec{v} \perp = (y, -x)
\]
Triangle Area

Given a triangle \( T = \{ p_1, p_2, p_3 \} \), the area of the triangle is half the base times the height:

\[
2 \cdot |T| = \| p_2 - p_1 \| \cdot \left| \left\langle p_3 - p_2, \frac{(p_1 - p_2)\perp}{\| (p_1 - p_2)\perp \|} \right\rangle \right|
\]

\[
= \left| \left\langle p_3 - p_2, (p_1 - p_2)\perp \right\rangle \right|
\]

If we drop the absolute value, we get the signed area:

\[
2 \cdot |T| = \left\langle p_3 - p_2, (p_1 - p_2)\perp \right\rangle
\]

This is positive if the vertices are in CCW order.
Triangle Area

Given a triangle $T = \{p_1, p_2, p_3\}$, the area of the triangle is half the base times the height:

$$2 \cdot |T| = \langle p_3 - p_2, (p_1 - p_2)\perp \rangle$$

Unless otherwise noted, we will use $| \cdot |$ to denote the signed area.

If we drop the absolute value, we get the signed area:

$$2 \cdot |T| = \langle p_3 - p_2, (p_1 - p_2)\perp \rangle$$

This is positive if the vertices are in CCW order.
Triangle Area

\[ 2 \cdot |T| = \langle p_3 - p_2, (p_1 - p_2)^\perp \rangle \]

Setting \( p_i = (x_i, y_i) \), this gives:

\[ 2 \cdot |T| = \langle (x_3 - x_2, y_3 - y_2), (y_1 - y_2, x_2 - x_1) \rangle \]

\[ = \sum_{i=1}^{3} (x_{i+1} + x_i) \cdot (y_{i+1} - y_i) \]
Triangle Area

\[ 2 \cdot |T| = \sum_{i=1}^{3} (x_{i+1} + x_i) \cdot (y_{i+1} - y_i) \]

Note:

If \( p_1 \) is at the origin, then the area becomes:

\[ 2 \cdot |T| = (x_3 + x_2) \cdot (y_3 - y_2) \]
Polygon Area (Take 1)

Triangulate the polygon and compute the sum of the triangle areas.

☒ Solving a harder problem than is required.
☒ Restricted to “simple” polygons.
☒ Doesn’t extend to higher dimensions.
Polygon Area (Take 2)

Divergence Theorem:

Let $V$ be a region in space with boundary $\partial V$, and let $\vec{F}$ be a vector field on $V$, then:

$$\int_V \text{div}(\vec{F}) = \int_{\partial V} \langle \vec{F}, \vec{N} \rangle$$

with $\vec{N}$ the normal on the boundary.
Polygon Area (Take 2)

Divergence Theorem:
\[ \int_{V} \text{div}(\vec{F}) = \int_{\partial V} \langle \vec{F}, \vec{N} \rangle \]

Taking \( \vec{F}(x, y) = (x, y) \), gives:
\[ 2 \int_{V} 1 = \int_{(x,y) \in \partial V} \langle (x, y), \vec{N} \rangle \]

\[ 2 \cdot |V| = \int_{\partial V} \langle p, \vec{N} \rangle \, dp \]
Polygon Area (Take 2)

\[ 2 \cdot |V| = \int_{\partial V} \langle p, \vec{N} \rangle dp \]

For a polygon \( P = \{p_1, ..., p_n\} \), we have:

\[ 2 \cdot |P| = \sum_{i=1}^{n} \int_{0}^{1} \langle (1 - t) \cdot p_i + t \cdot p_{i+1}, \vec{n}_i \rangle \cdot \|p_{i+1} - p_i\| \cdot dt \]

\[ = \sum_{i=1}^{n} \frac{1}{2} \cdot \langle p_i + p_{i+1}, \vec{n}_i \rangle \cdot \|p_{i+1} - p_i\| \]
Polygon Area (Take 2)

\[ 2 \cdot |P| = \sum_{i=1}^{n} \frac{1}{2} \cdot \langle p_i + p_{i+1}, \vec{n}_i \rangle \cdot \|p_{i+1} - p_i\| \]

Writing the normal as the $90^\circ$ rotation of the difference (normalized):

\[ \vec{n}_i = \frac{(p_{i+1} - p_i)^\perp}{\| (p_{i+1} - p_i)^\perp \|} = \frac{(p_{i+1} - p_i)^\perp}{\| p_{i+1} - p_i \|} \]
Polygon Area (Take 2)

\[ 2 \cdot |P| = \sum_{i=1}^{n} \frac{1}{2} \cdot \langle p_i + p_{i+1}, \vec{n}_i \rangle \cdot \|p_{i+1} - p_i\| \]

Writing the normal as the $90^\circ$ rotation of the difference (normalized):

\[ \vec{n}_i = \frac{(p_{i+1} - p_i)^\perp}{\| (p_{i+1} - p_i)^\perp \|} = \frac{(p_{i+1} - p_i)^\perp}{\| p_{i+1} - p_i \|} \]

\[ 2 \cdot |P| = \sum_{i=1}^{n} \frac{1}{2} \cdot \langle (p_{i+1} + p_i), (p_{i+1} - p_i)^\perp \rangle \]
Polygon Area (Take 2)

\[ 2 \cdot |P| = \sum_{i=1}^{n} \frac{1}{2} \cdot \langle (p_{i+1} + p_i), (p_{i+1} - p_i)\rangle \]

Noting that \((x, y)\)\(\perp = (y, -x)\) and writing \(p_i = (x_i, y_i)\), we get:

\[ 2 \cdot |P| = \sum_{i=1}^{n} (x_{i+1} + x_i) \cdot (y_{i+1} - y_i) \]
Polygon Area (Take 2)

\[ 2 \cdot |P| = \sum_{i=1}^{n} \frac{1}{2} \cdot \langle (p_{i+1} + p_i) , (p_{i+1} - p_i)^\perp \rangle \]

Computing the area of a polygon requires two adds and one multiply per vertex.

\[ 2 \cdot |P| = \sum_{i=1}^{n} (x_{i+1} + x_i) \cdot (y_{i+1} - y_i) \]
Polygon Area (Take 2)

\[ 2 \cdot |P| = \sum_{i=1}^{n} (x_{i+1} + x_i) \cdot (y_{i+1} - y_i) \]

Q: What’s really going on?

A: For a triangle \( \{p_1, p_2, p_3\} \), if \( p_1 \) is at the origin, the area is:

\[ 2 \cdot |T| = (x_3 + x_2) \cdot (y_3 - y_2) \]
Polygon Area (Take 2)

\[ 2 \cdot |P| = \sum_{i=1}^{n} (x_{i+1} + x_i) \cdot (y_{i+1} - y_i) \]

Q: What’s really going on?

A: Sum the areas of the triangles defined by the origin and the polygon edges.
Polygon Area (Take 2)

\[2 \cdot |P| = \sum_{i=1}^{n} (x_{i+1} + x_i) \cdot (y_{i+1} - y_i)\]

Q: What’s really going on?

A: Sum the areas of the triangles defined by the origin and the polygon edges.
Polygon Area (Take 2)

\[ 2 \cdot |P| = \sum_{i=1}^{n} (x_{i+1} + x_i) \cdot (y_{i+1} - y_i) \]

Q: What’s really going on?

A: Sum the areas of the triangles defined by the origin and the polygon edges.
Polygon Area (Take 2)

\[ 2 \cdot |P| = \sum_{i=1}^{n} (x_{i+1} + x_i) \cdot (y_{i+1} - y_i) \]

Q: What’s really going on?

A: Sum the areas of the triangles defined by the origin and the polygon edges.
Polygon Area (Take 2)

\[ 2 \cdot |P| = \sum_{i=1}^{n} (x_{i+1} + x_i) \cdot (y_{i+1} - y_i) \]

Q: What’s really going on?

A: Sum the areas of the triangles defined by the origin and the polygon edges.
Polygon Area (Take 2)

\[ 2 \cdot |P| = \sum_{i=1}^{n} (x_{i+1} + x_i) \cdot (y_{i+1} - y_i) \]

Q: What’s really going on?

A: Sum the areas of the triangles defined by the origin and the polygon edges.
 Polygon Area (Take 2)

\[ 2 \cdot |P| = \sum_{i=1}^{n} (x_{i+1} + x_i) \cdot (y_{i+1} - y_i) \]

Q: What’s really going on?

A: Sum the areas of the triangles defined by the origin and the polygon edges.
Polygon Area (Take 2)

\[ 2 \cdot |P| = \sum_{i=1}^{n} (x_{i+1} + x_i) \cdot (y_{i+1} - y_i) \]

Q: What’s really going on?

A: Sum the areas of the triangles defined by the origin and the polygon edges.
Polygon Area (Take 2)

\[ 2 \cdot |P| = \sum_{i=1}^{n} (x_{i+1} + x_i) \cdot (y_{i+1} - y_i) \]

Q: What’s really going on?

A: Sum the areas of the triangles defined by the origin and the polygon edges.
Polygon Area (Take 2)

\[2 \cdot |P| = \sum_{i=1}^{n} (x_{i+1} + x_i) \cdot (y_{i+1} - y_i)\]

Q: What’s really going on?

A: Sum the areas of the triangles defined by the origin and the polygon edges.
Polygon Area (Take 2)

$$2 \cdot |P| = \sum_{i=1}^{n} (x_{i+1} + x_i) \cdot (y_{i+1} - y_i)$$

Q: What’s really going on?

A: Sum the areas of the triangles defined by the origin and the polygon edges.
2 \cdot |P| = \sum_{i=1}^{n} (x_{i+1} + x_i) \cdot (y_{i+1} - y_i)

In this “triangulation”, the use of signed area cancels out the unwanted contribution.

A: Sum the areas of the triangles defined by the origin and the polygon edges.
Polygon Area (Take 2)

Note:

The same approach can be used to compute the volume enclosed by a triangle mesh in 3D:

- Pick a base point.
- Create tetrahedra by joining the base point to the triangles of the mesh.
- Sum the signed volumes of the tetrahedra.
Outline

• Polygon Area

• Implementation
// A general structure for points with integer coordinates in arbitrary dimensions

template< unsigned int D >
struct Point
{
    int c[D];
    Point( void ){ memset( c, 0, sizeof(int)*D ); }
    int &operator[]( int idx ) { return c[idx]; }
    int  operator[]( int idx ) const { return c[idx]; }
};
Implementation

long long Area2( Point< 2 > p0 , Point< 2 > p1 , Point< 2 > p2 );
{
    long long a = 0;
    a += ( (long long)( p1[0] + p0[0] ) ) * ( p1[1] - p0[1] );
    a += ( (long long)( p2[0] + p1[0] ) ) * ( p2[1] - p1[1] );
    a += ( (long long)( p0[0] + p2[0] ) ) * ( p0[1] - p2[1] );
    return a;
}
Implementation

// A circular linked-list structure for representing a vertex
// within a polygon in 2D
struct PVertex
{
    Point< 2 > p;
    PVertex *prev, *next;
    PVertex( Point< 2 > _p );
    PVertex &addBefore( Point< 2 > p );
    unsigned int size( void ) const;
    long long area2( void ) const;
    static PVertex *Remove( PVertex *v );
};
Implementation

PVertex::PVertex( Point< 2 > _p ){ p=_p , prev = next = this; }
Implementation

PVertex& PVertex::addBefore( Point< 2 > p )
{
    PVertex *v = new PVertex(p);
    v->prev = prev, v->next = this;
    prev->next = v;
    prev = v;
    return *v;
};
Implementation

static PVertex *PVertex::Remove( PVertex *v )
{
    PVertex *temp = v->prev;
    v->prev->next = v->next;
    v->next->prev = v->prev;
    delete v;
    return temp==v ? NULL : temp;
}
Implementation

unsigned int PVertex::size( void ) const
{
    unsigned int s = 0;
    for( const PVertex *v=this ; ; v=v->next )
    {
        s++;
        if( v->next==this ) break;
    }
    return s;
}
Implementation

long long PVertex::area2( void ) const
{
    long long a = 0;
    for( const PVertex *v=this ; ; v=v->next )
    {
        a += Area2( Point< 2 >(), v->p , v->next->p );
        if( v->next==this ) break;
    }
    return a;
}
Sidedness

Given a line segment, $\overrightarrow{pq}$, and a point $r$, we can determine if $r$ is to the left of, on, or to the right of $\overrightarrow{pq}$ by testing the sign of the area of triangle $\Delta pqr$.
bool Left( Point<2> p, Point<2> q, Point<2> r )
{ return Area2( p, q, r ) > 0; }

bool LeftOn( Point<2> p, Point<2> q, Point<2> r )
{ return Area2( p, q, r ) >= 0; }

bool Collinear( Point<2> p, Point<2> q, Point<2> r )
{ return Area2( p, q, r ) == 0; }

bool Right( Point<2> p, Point<2> q, Point<2> r )
{ return Area2( p, q, r ) < 0; }

bool RightOn( Point<2> p, Point<2> q, Point<2> r )
{ return Area2( p, q, r ) <= 0; }
Point on Line Segment

Given a line segment, $\overline{pq}$, a point $r$ is between $p$ and $q$ if:

- $r$ is on the line between $p$ and $q$, and
- the $x$-coordinate of $r$ is between the $x$-coordinates of $p$ and $q$
Point on Line Segment

Given a line segment, $\overline{pq}$, a point $r$ is between $p$ and $q$ if:

- $r$ is on the line between $p$ and $q$, and
- the $x$-coordinate of $r$ is between the $x$-coordinates of $p$ and $q$ (if $\overline{pq}$ is not vertical)
- the $y$-coordinate of $r$ is between the $y$-coordinates of $p$ and $q$ (if $\overline{pq}$ is vertical)
Implementation

```cpp
bool Between( Point<2> p , Point<2> q , Point<2> r )
{
    if( !Collinear( p , q , r ) ) return false;
    unsigned int dir = p[0]!=q[0] ? 0 : 1;
            ( q[dir] <= r[dir] && r[dir] <= p[dir] );
}
```
Proper Intersection

Line segments $\overline{pq}$ and $\overline{rs}$, intersect properly if they intersect in their interior:

- Neither $r$ nor $s$ is on the segment $\overline{pq}$.
- Neither $p$ nor $q$ is on the segment $\overline{rs}$.
- $p$ and $q$ are on different sides of $\overline{rs}$, and $r$ and $s$ are on different sides of $\overline{pq}$.
Implementation

```cpp
bool IsectProper( Point<2> p, Point<2> q, Point<2> r, Point<2> s )
{
    if( Collinear( p, q, r ) || Collinear( p, q, s ) ) return false;
    if( Collinear( r, s, p ) || Collinear( r, s, q ) ) return false;
    if( Left( p, q, r ) == Left( p, q, s ) ) return false;
    if( Left( r, s, p ) == Left( r, s, q ) ) return false;
    return true;
}
```
Intersection

Line segments $\overline{pq}$ and $\overline{rs}$, intersect if:

- $p$ is between $r$ and $s$, or
- $q$ is between $r$ and $s$, or
- $r$ is between $p$ and $q$, or
- $s$ is between $p$ and $q$, or
- they intersect properly.
Implementation

bool Isect( Point<2> p, Point<2> q, Point<2> r, Point<2> s )
{
    return
        IsectProper( p, q, r, s ) ||
        Between( p, q, r ) || Between( p, q, s ) ||
        Between( r, s, p ) || Between( r, s, q );
}

✓

✓

✗

✗

✓
Diagonal

Property:

Given a polygon, \( P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2 \), an edge \( \overline{p_i p_j} \) is a diagonal if:

1. \( \forall p_k \in P \text{ w/ } k, k + 1 \notin \{i, j\}: \overline{p_i p_j} \cap \overline{p_k p_{k+1}} = \emptyset \)
2. \( \overline{p_i p_j} \) is internal to \( P \) around \( p_i \) and \( p_j \)
Edge Intersection

To test the first property:

1. \( \forall p_k \in P \text{ w/ } k, k + 1 \notin \{i, j\}: p_ip_j \cap p_kp_{k+1} = \emptyset \)

we check for the intersection of \( p_ip_j \) with all edges.
Implementation

```cpp
bool DiagonalIsect( const PVertex< 2 > *r , const PVertex< 2 > *s )
{
    for( const PVertex< 2 > *v=r ; v!=s ; v=v->next )
    {
        if( v->prev!=r && v->prev!=s && v!=r && v!=s )
            if( Isect( r->p , s->p , v->prev->p , v->p ) ) return true;
        if( v->next==r ) break;
    }
    return false;
}
```

Complexity: $O(n)$
Cone Interior

Given points $p$, $q$, and $r$, a line segment $\overline{qs}$ is *in the cone of $pqr$* if $\overline{qs}$ is strictly interior to the region swept out CW from $\overrightarrow{qp}$ to $\overrightarrow{qr}$.

- If $\angle pqr$ is a left turn (i.e. $q$ is convex):
  - $s$ must be to the left of both $\overrightarrow{pq}$ and $\overrightarrow{qr}$.
- Otherwise:
  - $s$ cannot be to the right of or on both $\overrightarrow{pq}$ and $\overrightarrow{qr}$. 
Implementation

```cpp
bool InCone(Point<2> p, Point<2> q, Point<2> r, Point<2> s)
{
    if (Left(p, q, r))
        return (Left(p, q, s) && Left(q, r, s));
    else
        return !(RightOn(p, q, s) && RightOn(q, r, s));
}
```
Implementation

```cpp
bool InCones( const PVertex< 2 >* r , const PVertex< 2 >* s )
{
    return
         InCone( r->prev->p , r->p , r->next->p , s->p ) &&
         InCone( s->prev->p , s->p , s->next->p , q->p );
}
```

Complexity: O(1)
Implementation

```cpp
bool IsDiagonal( const PVertex< 2 >* r, const PVertex< 2 >* s )
{
    return InCones( r, s ) && !DiagonalIsect( r, s );
}
```

Complexity: $O(n)$
Trangulation (Naïve)

Recursively:

1. If the polygon is a triangle, output the triangle.
2. Otherwise
   a. Find diagonal.
   b. Split the polygon in two.
Implementation

void OutputTriangulation( PVertex< 2 > *poly )
{
    if( poly->size()>3 )
    {
        PVertex< 2 > *r , *s , *poly1 , *poly2;
        GetDiagonal( poly , r , s )
        SplitOnDiagonal( poly , r , s , poly1 , poly2 );
        OutputTriangulation( poly1 );
        OutputTriangulation( poly2 );
    }
    else Output( poly );
}
Triangulation (Ear Removal)

While there are more than three vertices:

1. Find an ear $p_i$.
2. Output the triangle $\{p_{i-1}, p_i, p_{i+1}\}$.
3. Remove $p_i$ from the polygon.

Note:
The ear status can only change for the vertices $p_{i-1}$ and $p_{i+1}$. 
Triangulation (Ear Removal)

Initialize the ear status of all vertices.

While there are more than three vertices:

1. Find an ear $p_i$.
2. Output the triangle $\{p_{i-1}, p_i, p_{i+1}\}$.
3. Remove $p_i$ from the polygon.
4. Update the ear status of $p_{i-1}$ and $p_{i+1}$.
Implementation

// Assumes member:
//
//   bool PVertex< 2 >::isEar

bool InitEars( PVertex< 2 > *poly )
{
    for( PVertex< 2 > *v=poly ; ; v=v->next )
    {
        v->isEar = IsDiagonal( poly , v->prev , v->next );
        if( v->next==poly ) break;
    }
}

Complexity: \( O(n^2) \)
Implementation

PVertex< 2 > *ProcessEar( PVertex< 2 > *e )
{
    Output( e->prev , e , e->next );
    e->prev->isEar = IsDiagonal( e->prev->prev , e->next );
    e->next->isEar = IsDiagonal( e->prev , e->next->next );
    return PVertex< 2 >::Remove( e );
}

Complexity: O(n)
Implementation

```c
void OutputTriangulation(PVertex< 2 > *poly )
{
    InitEars( poly );
    unsigned int sz = poly->size();
    while( sz>3 )
    {
        for( PVertex< 2 > *v=poly ; ; v=v->next )
        {
            if( v->isEar ){ poly = ProcessEar( v ) ; sz-- ; break; }
            if( v->next==poly ) break;
        }
    }
}
```

Complexity: \( O(n^2) \)