Implementation Details

Lecture 7 - February 6, 2009 - 1-3 PM

Outline

Outline

- Data Structure
- Algorithms
- Conclusions
- Suggested Readings
- Acknowledgments

Data Structure

Data Structure

We can use any "decent" existing data structure for surface meshes.

Data Structure

We can use any "decent" existing data structure for surface meshes.

Basic elements:

Data Structure

We can use any “decent” existing data structure for surface meshes.

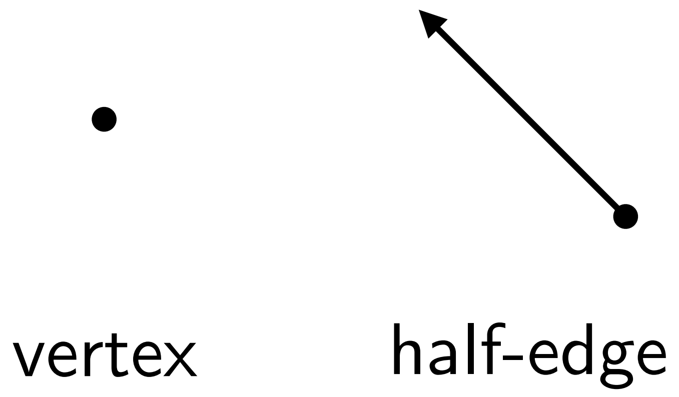
Basic elements:

vertex

Data Structure

We can use any "decent" existing data structure for surface meshes.

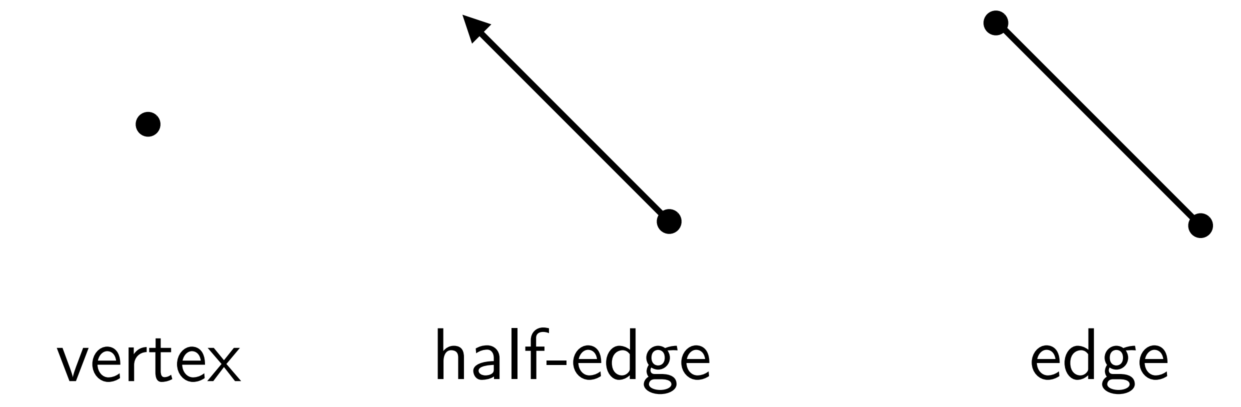
Basic elements:



Data Structure

We can use any "decent" existing data structure for surface meshes.

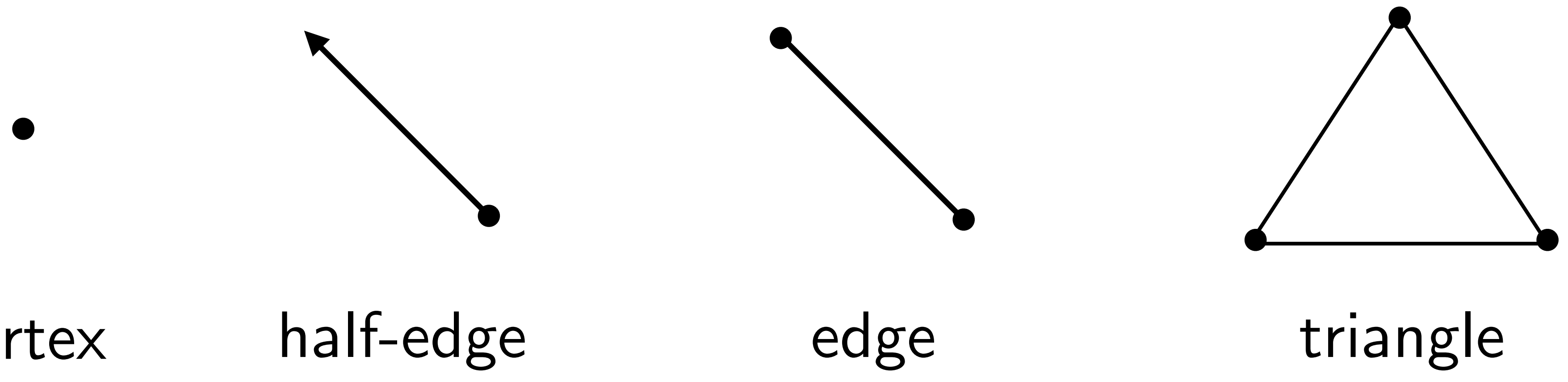
Basic elements:



Data Structure

We can use any “decent” existing data structure for surface meshes.

Basic elements:



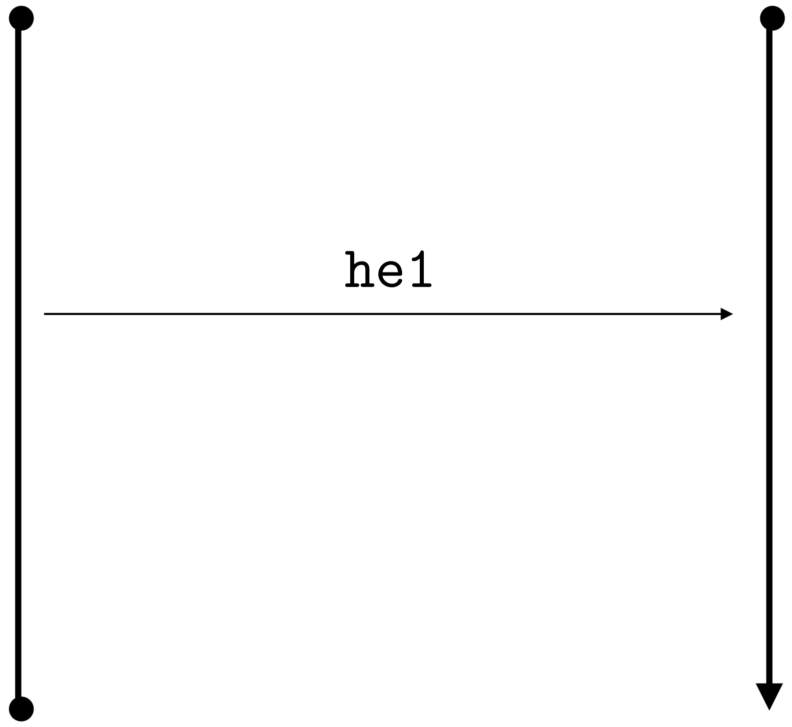
Data Structure

Data Structure

- Edge

Data Structure

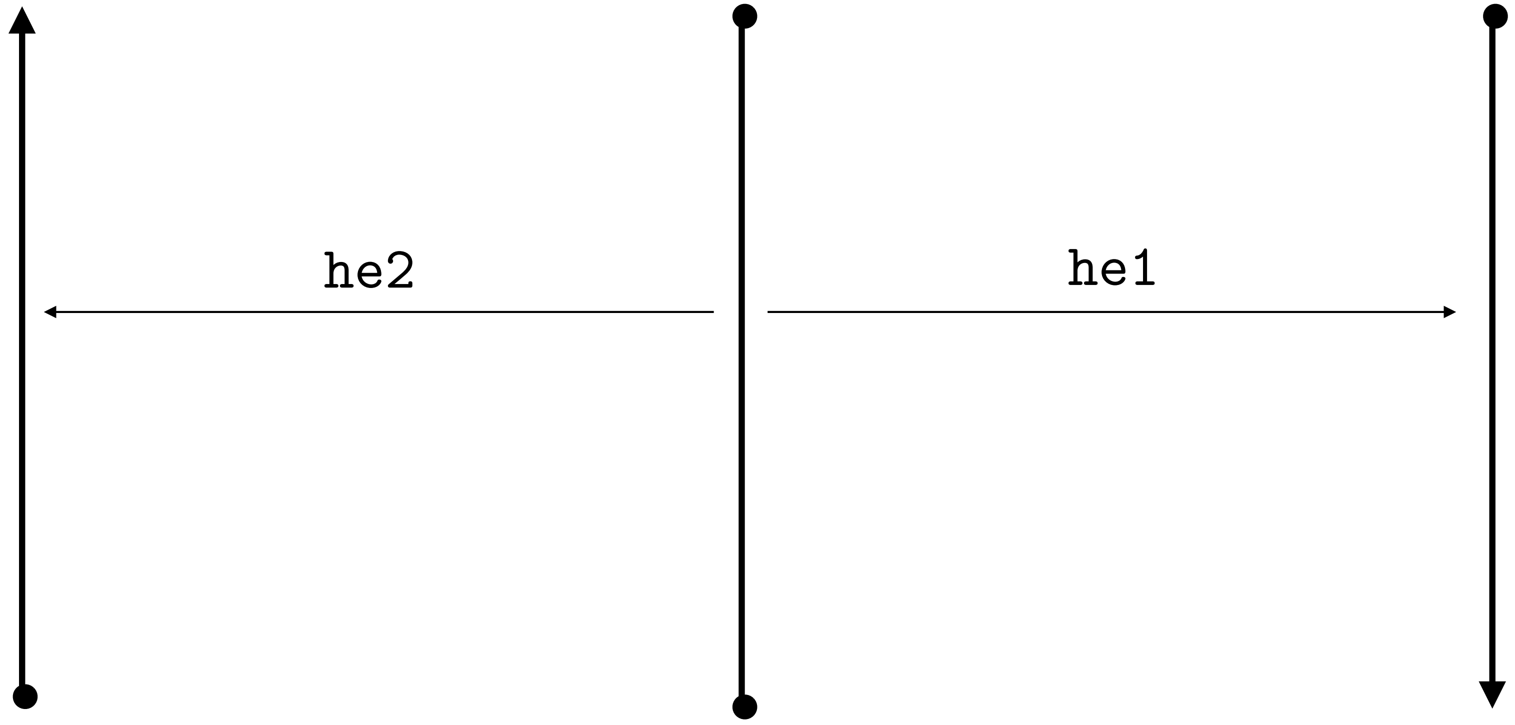
- Edge



Pointer to half-edge 1

Data Structure

- Edge

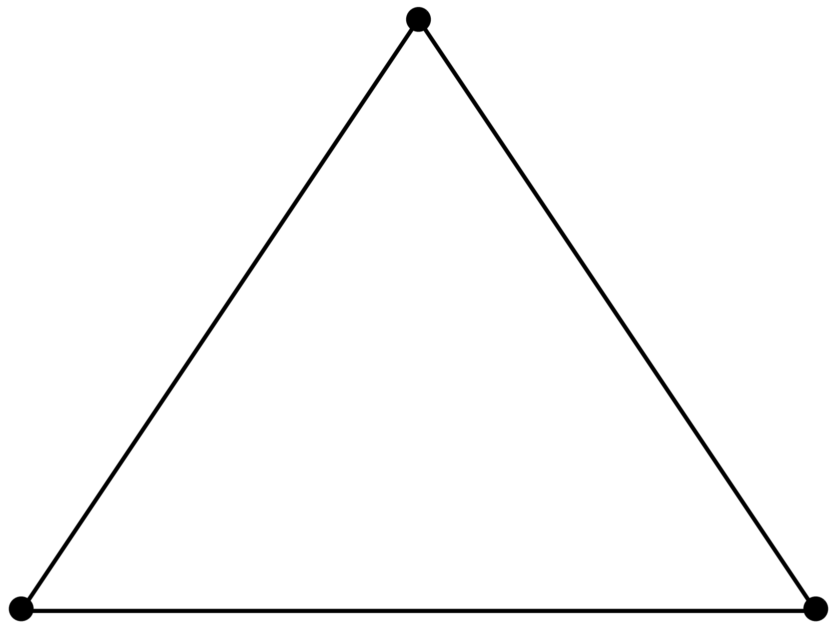


Pointer to half-edge 2 Pointer to half-edge 1

Data Structure

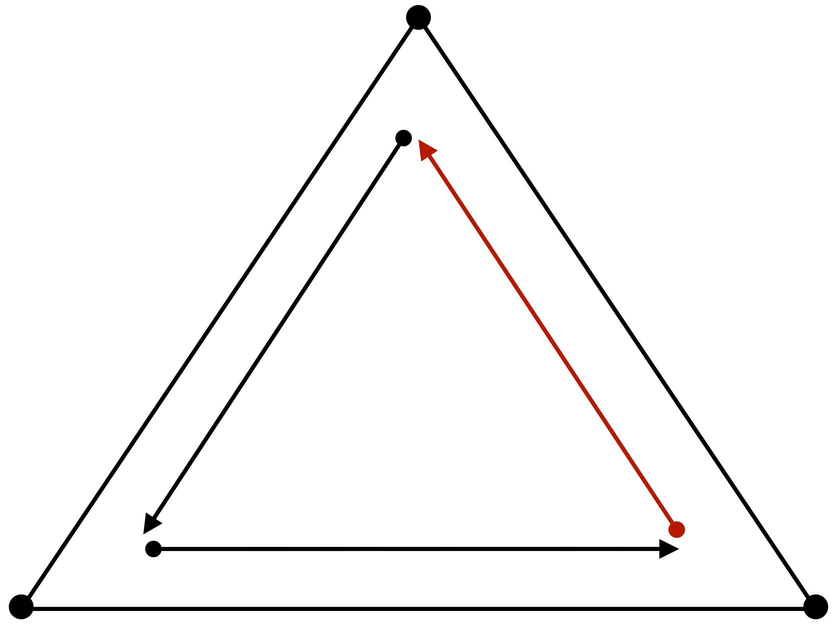
Data Structure

- Triangle



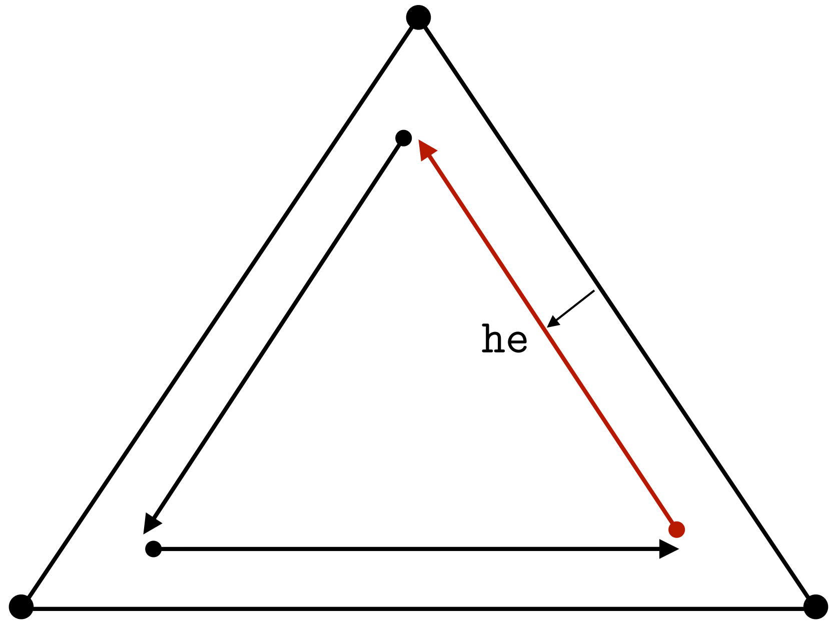
Data Structure

- Triangle



Data Structure

- Triangle



Pointer to one half-edge (known as the *first*)

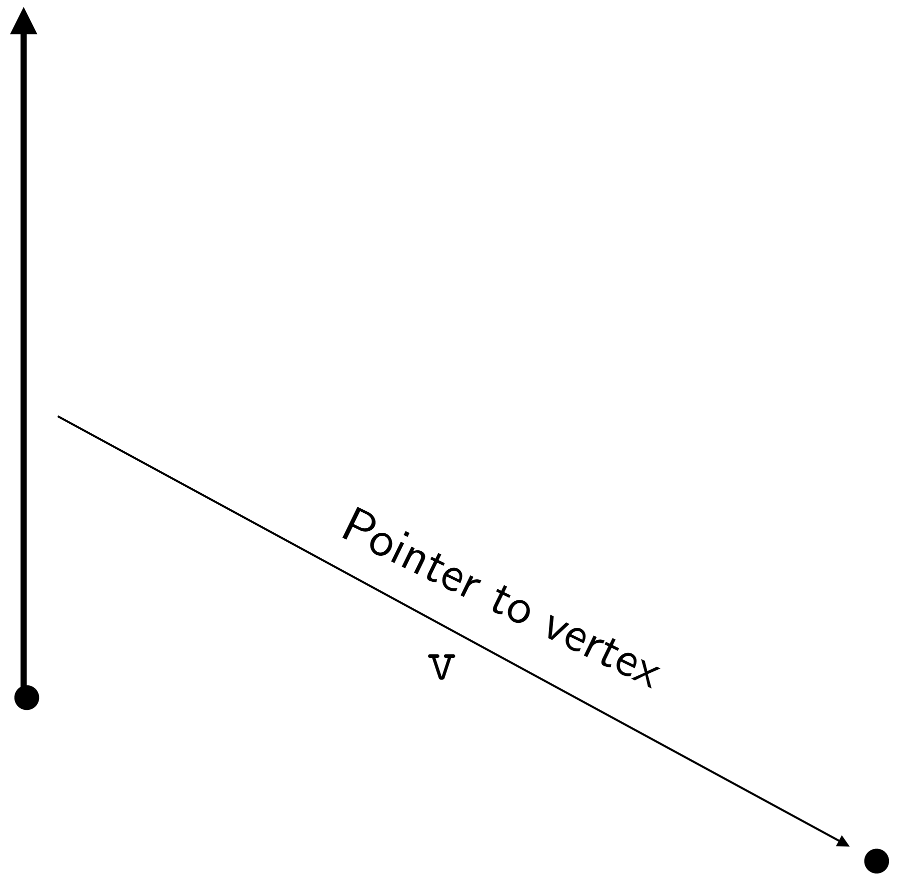
Data Structure

Data Structure

- Half-edge

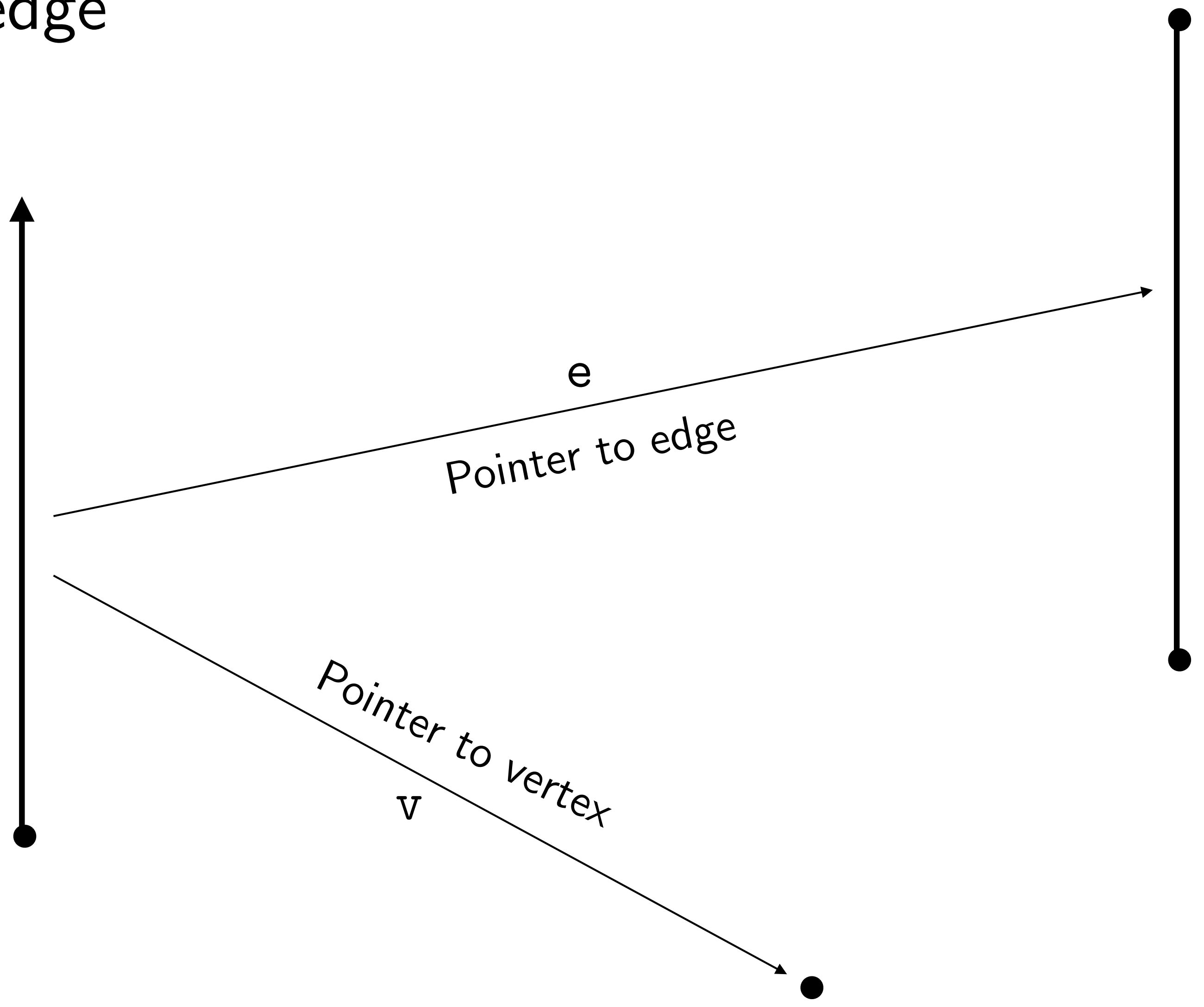
Data Structure

- Half-edge



Data Structure

- Half-edge



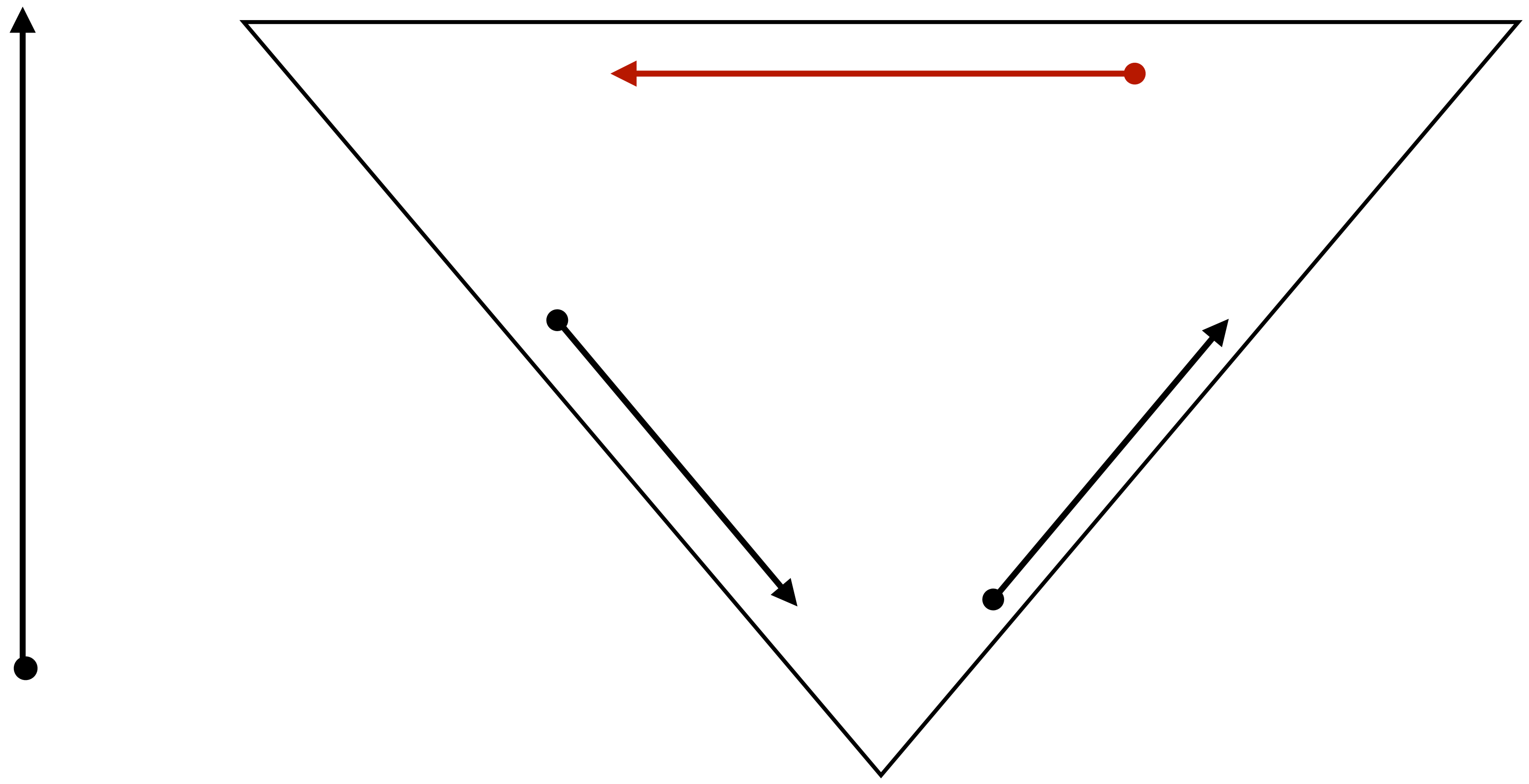
Data Structure

Data Structure

- Half-edge

Data Structure

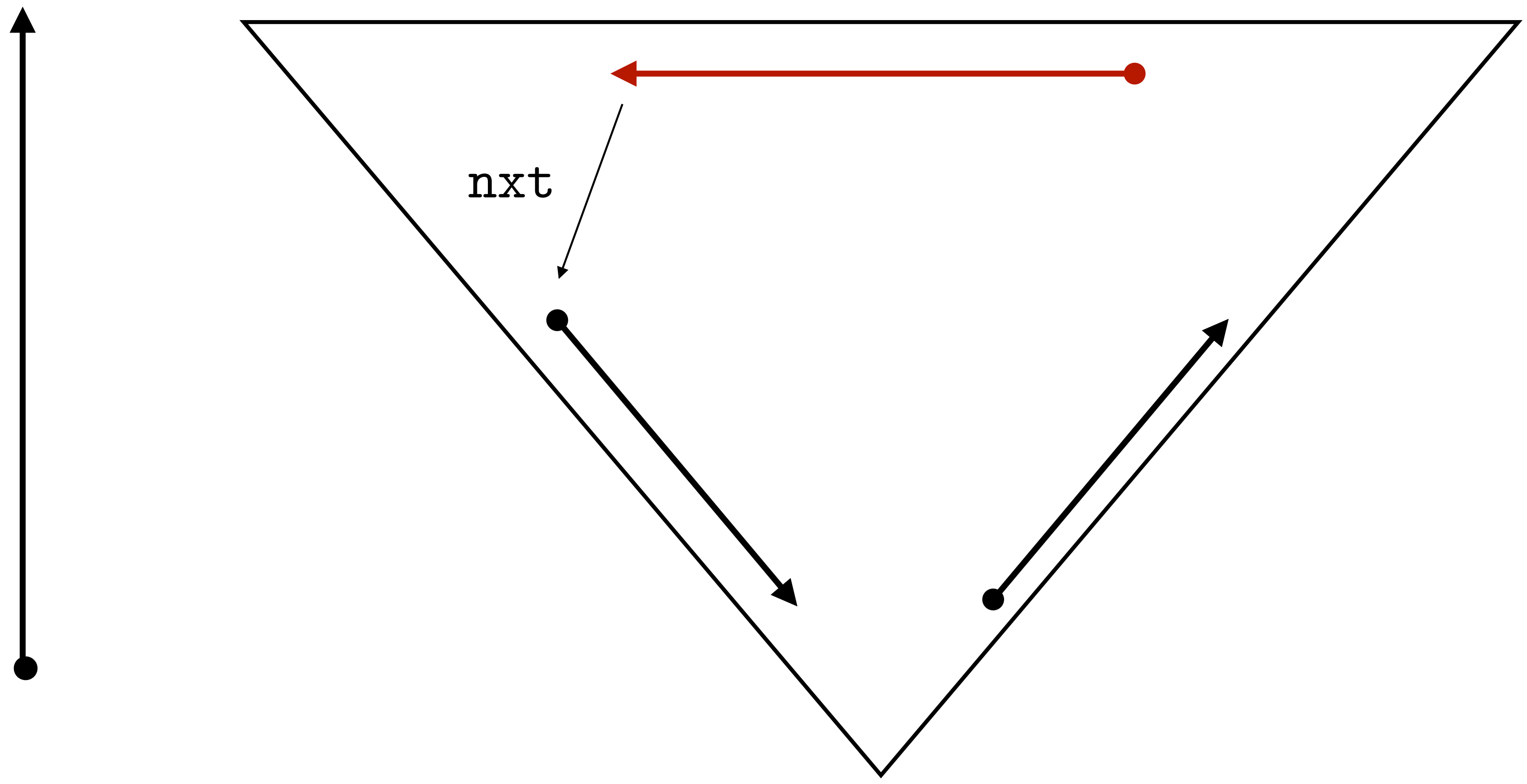
- Half-edge



Data Structure

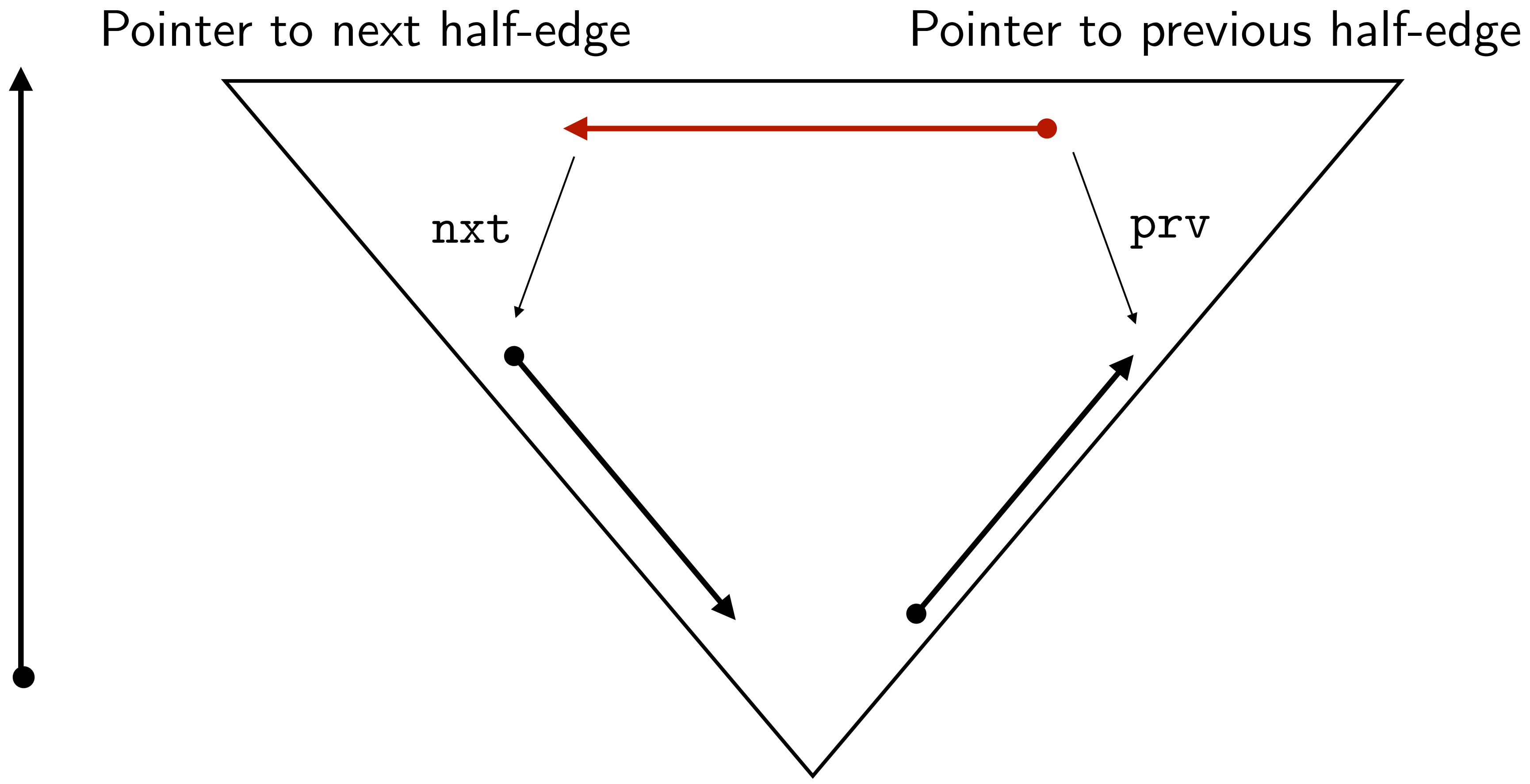
- Half-edge

Pointer to next half-edge



Data Structure

- Half-edge



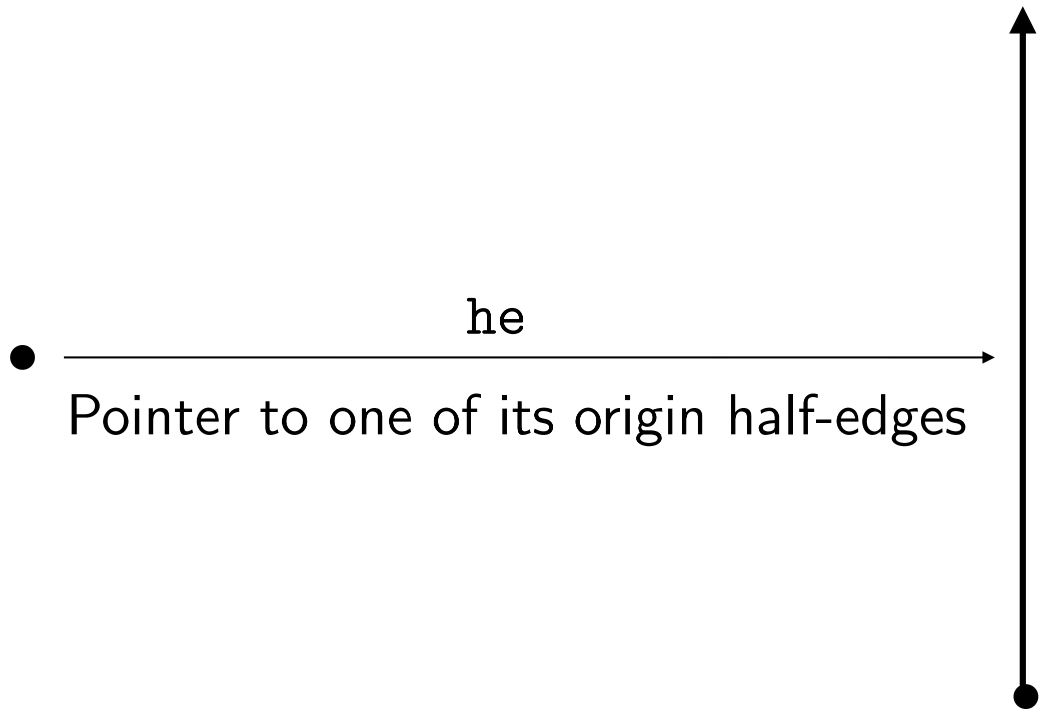
Data Structure

Data Structure

- Vertex
  -

Data Structure

- Vertex

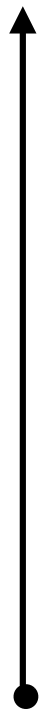


Data Structure

- Vertex

  - $\xrightarrow{\texttt{he}}$

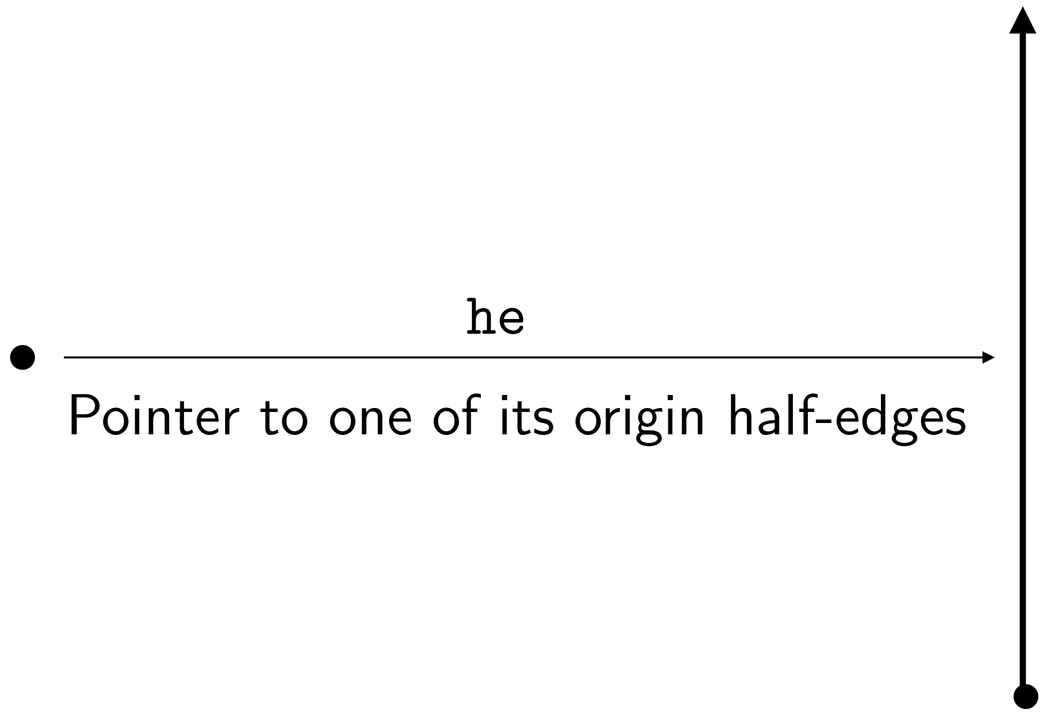
    Pointer to one of its origin half-edges



Any origin half-edge will do!

Data Structure

- Vertex



Any origin half-edge will do!

Coordinates, (x, y, z), of the vertex

Data Structure

Data Structure

- Mesh

Data Structure

- Mesh

  `ltris`: pointer to a list of all triangles

Data Structure

- Mesh

  `ltris`: pointer to a list of all triangles

  `ledgs`: pointer to a list of all edges

Data Structure

- Mesh

  `ltris`: pointer to a list of all triangles

  `ledgs`: pointer to a list of all edges

  `lvrts`: pointer to a list of all vertices

Data Structure

Data Structure

- Fundamental operations

Data Structure

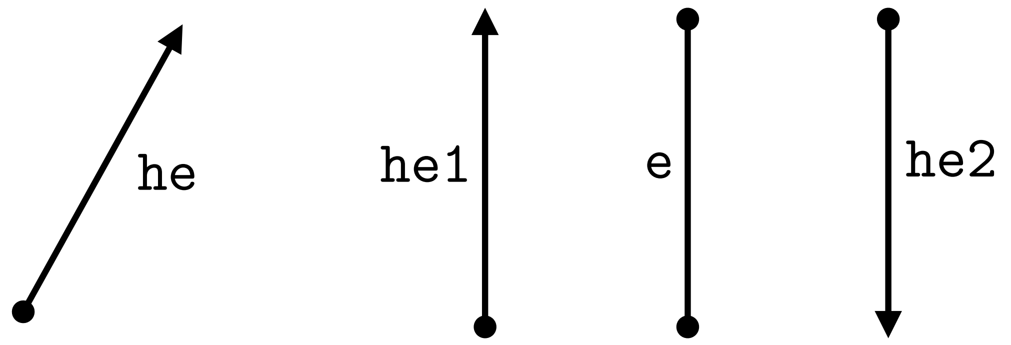
- Fundamental operations

  Given a pointer, `he`, to a half-edge, find its *mate*:

Data Structure

- Fundamental operations

  Given a pointer, `he`, to a half-edge, find its *mate*:

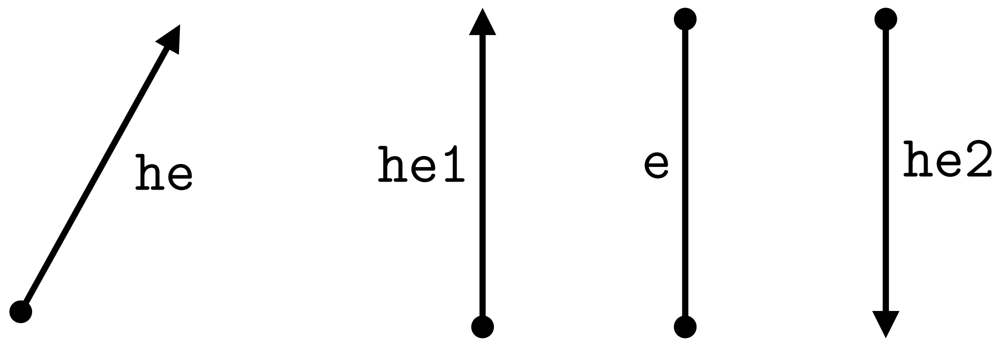


`he = he1 or he = he2?`

Data Structure

- Fundamental operations

  Given a pointer, `he`, to a half-edge, find its *mate*:



he = he1 or he = he2?

MATE(he)

```
return ( he->e->he1 == he ) ?  he->e->he2 ?  he->e->he1 ;
```

Data Structure

Data Structure

- Fundamental operations

Data Structure

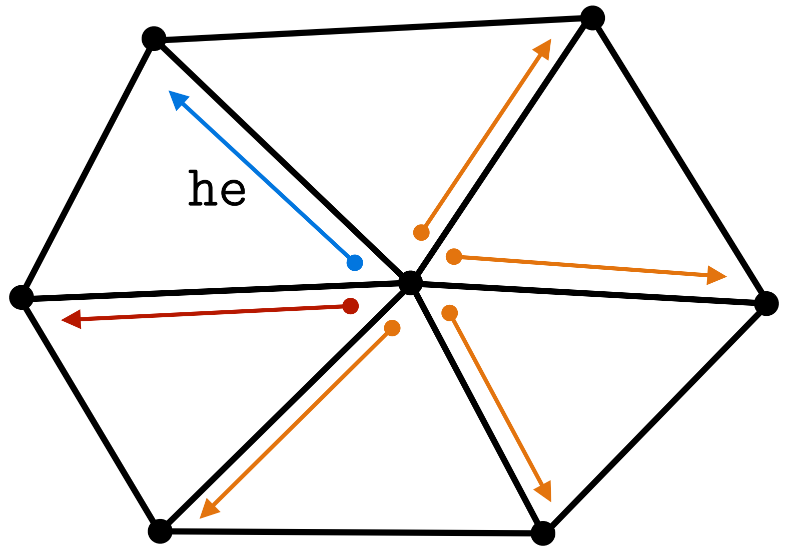
- Fundamental operations

  Given a pointer, `he`, to a half-edge, return a pointer to *successor* half-edge in a counterclockwise traverse of the cycle of half-edges with the same vertex origin as the half-edge pointed by `he`.

Data Structure

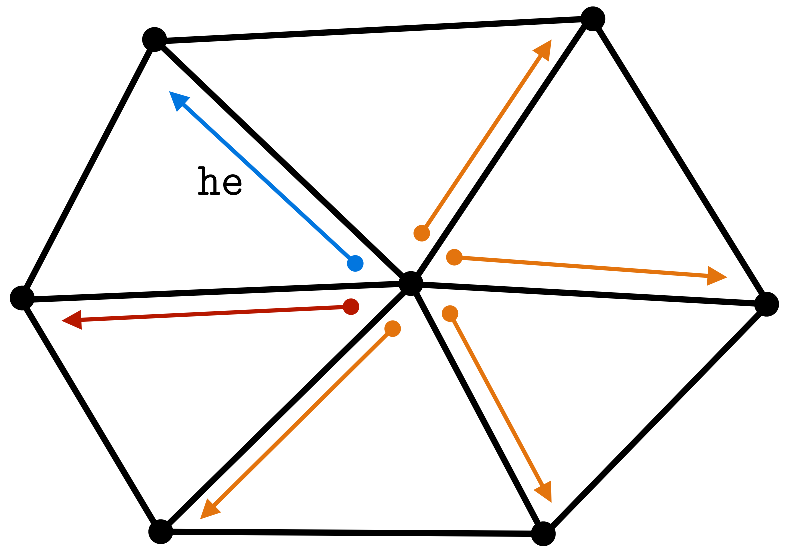
- Fundamental operations

  Given a pointer, `he`, to a half-edge, return a pointer to *successor* half-edge in a counterclockwise traverse of the cycle of half-edges with the same vertex origin as the half-edge pointed by `he`.

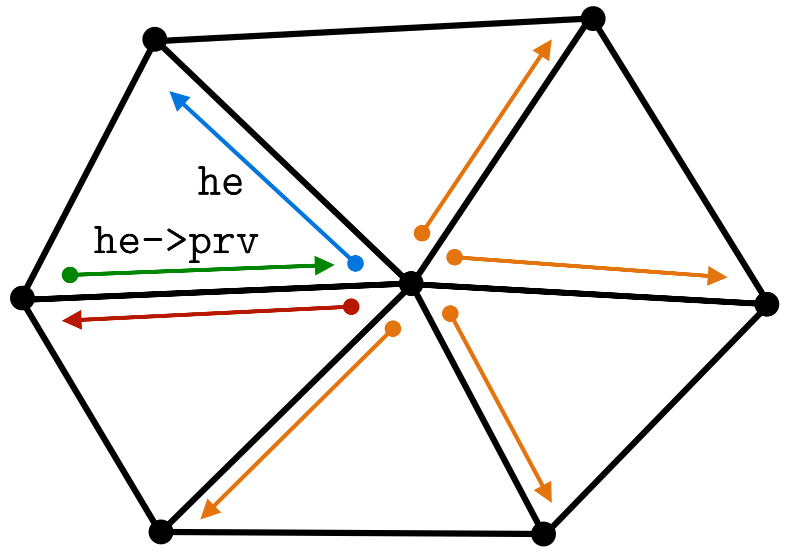


Data Structure

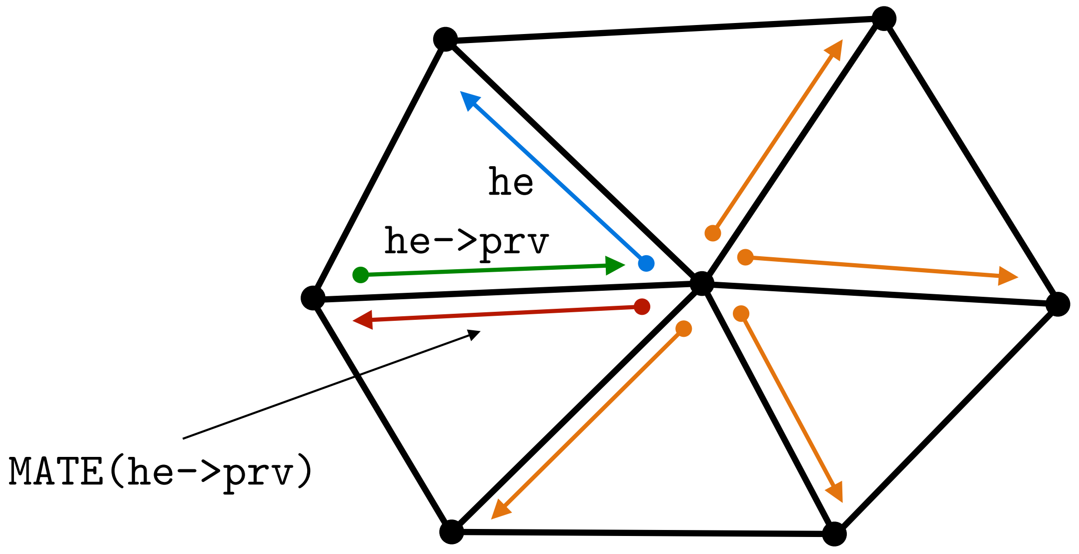
Data Structure



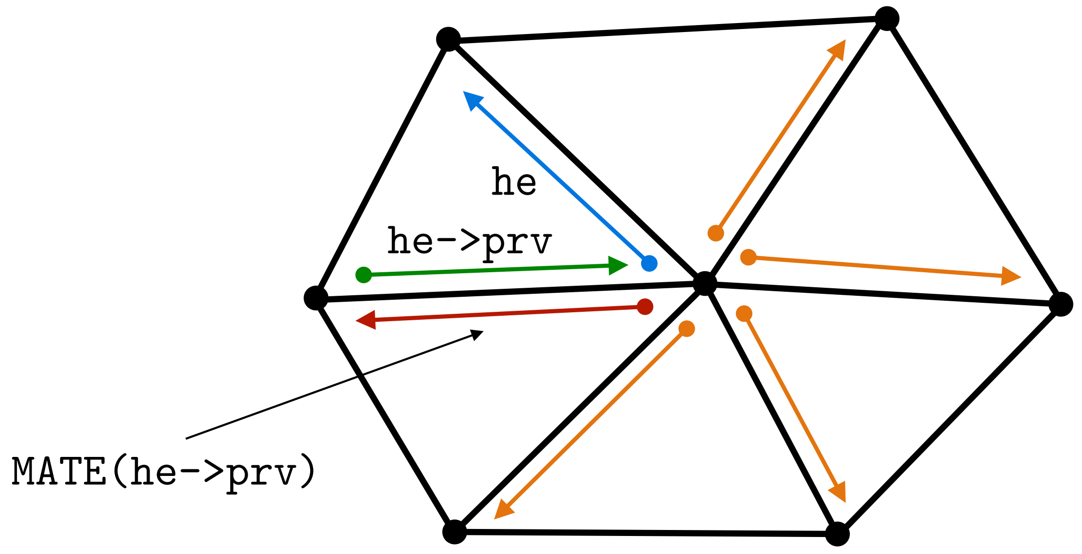
Data Structure



Data Structure



Data Structure



```
SUCC-CC(he)

return MATE(he->prv) ;
```

Data Structure

Data Structure

How can we augment the data structure to store the PPS information?

Data Structure

How can we augment the data structure to store the PPS information?

A good strategy (*from the software engineering point of view*) is to include a *generic* pointer to *attributes* in every basic element (vertex, half-edge, edge, and triangle) of the data structure.

Data Structure

Data Structure

Let v be a vertex.

Data Structure

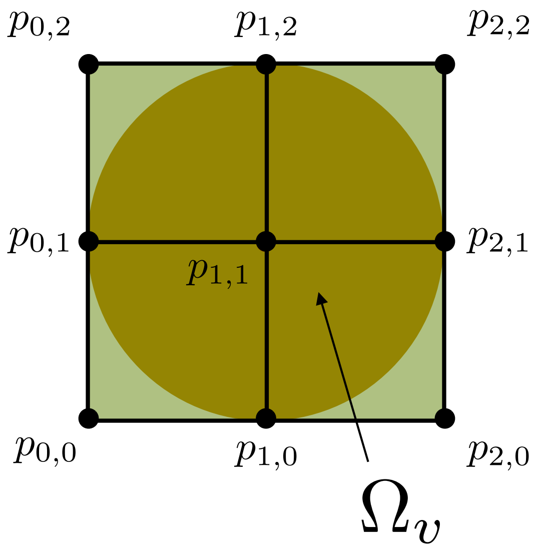
Let v be a vertex.

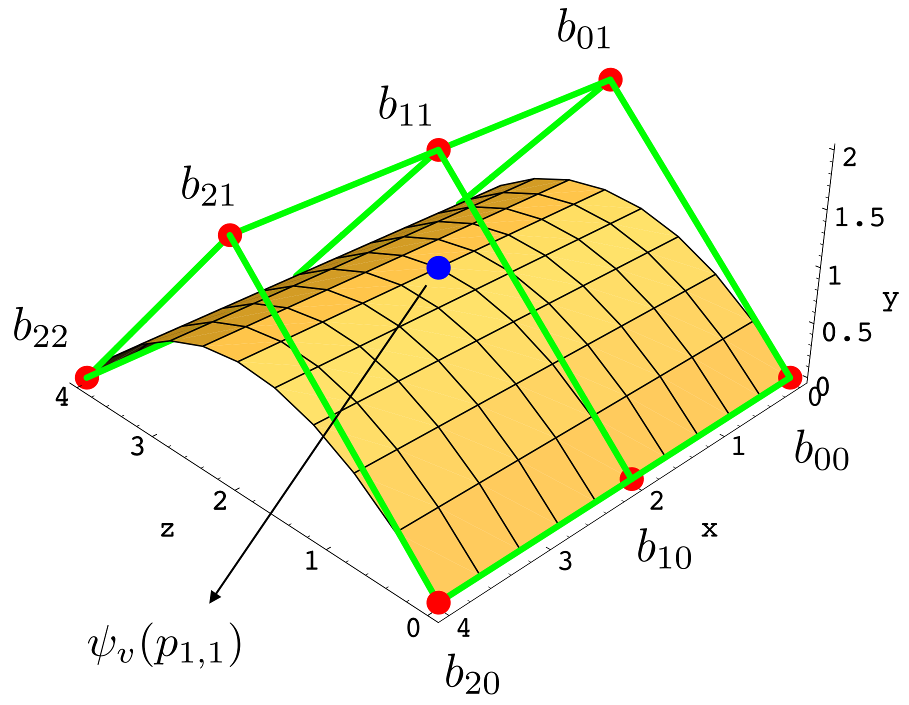
The only vertex attribute we really need is a pointer to a rectangular Bézier patch, which is the patch, ψ_v, associated with v.

Data Structure

Let v be a vertex.

The only vertex attribute we really need is a pointer to a rectangular Bézier patch, which is the patch, ψ_v, associated with v.





Data Structure

Data Structure

The rectangular Bézier patch itself is another *object* (separated from the mesh data structure) which stores the control points of the patch.

Data Structure

The rectangular Bézier patch itself is another *object* (separated from the mesh data structure) which stores the control points of the patch.

To speed up computations, we compute and store the degree, m_v, of v.

Data Structure

The rectangular Bézier patch itself is another *object* (separated from the mesh data structure) which stores the control points of the patch.

To speed up computations, we compute and store the degree, m_v, of v.

So, we have *two* vertex attributes: the Bézier patch and the vertex degree.

Data Structure

Data Structure

Let h be a half-edge.

Data Structure

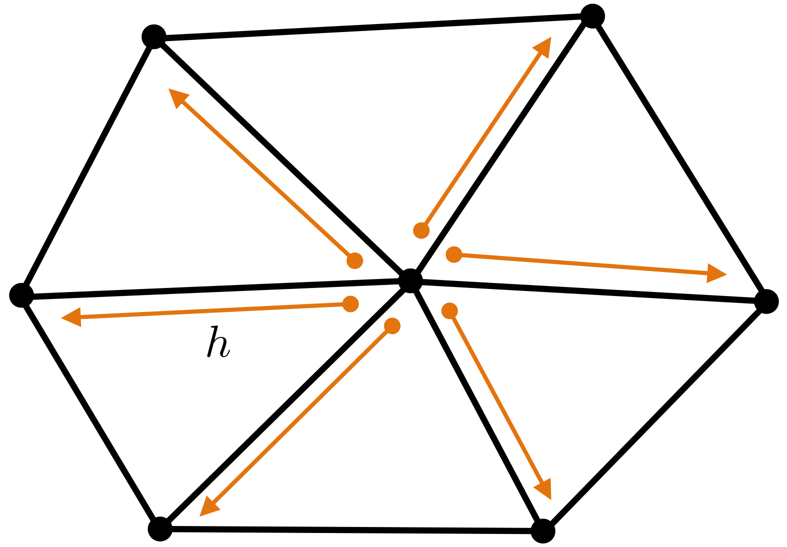
Let h be a half-edge.

The only attribute associated with h is its *id*, a number that uniquely identifies h in the cycle of half-edges sharing the same origin vertex as h.

Data Structure

Let h be a half-edge.

The only attribute associated with h is its *id*, a number that uniquely identifies h in the cycle of half-edges sharing the same origin vertex as h.



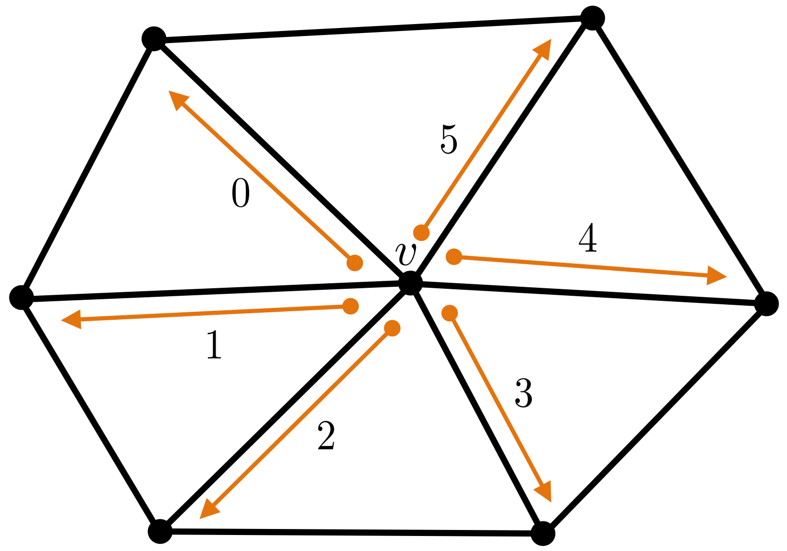
Data Structure

Data Structure

The *id* number is generated sequentially (using a counterclockwise traverse of the cycle of half-edges). The starting half-edge can be arbitrarily chosen. We choose the half-edge pointed by the vertex v.

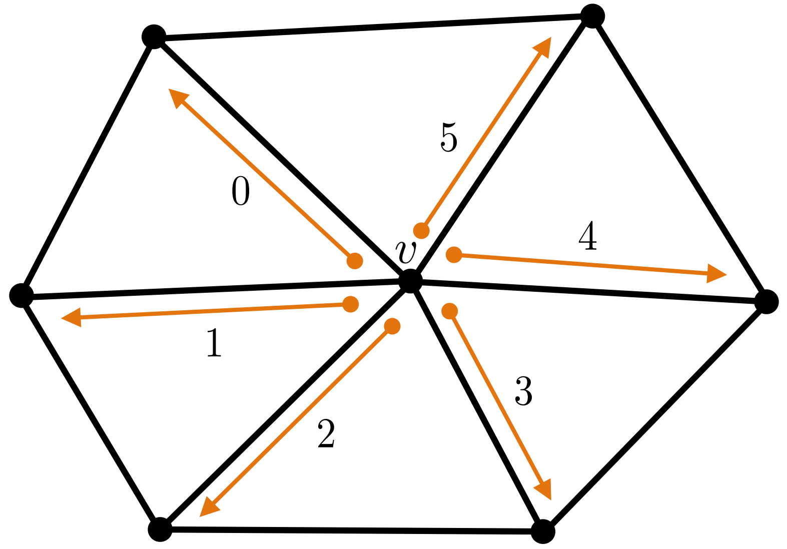
Data Structure

The *id* number is generated sequentially (using a counterclockwise traverse of the cycle of half-edges). The starting half-edge can be arbitrarily chosen. We choose the half-edge pointed by the vertex v.



Data Structure

The *id* number is generated sequentially (using a counterclockwise traverse of the cycle of half-edges). The starting half-edge can be arbitrarily chosen. We choose the half-edge pointed by the vertex v.



The *id* belongs to $\{0, \ldots, m_v - 1\}$, where m_v is the degree of v.

Data Structure

Data Structure

Why do we need this attribute?

Data Structure

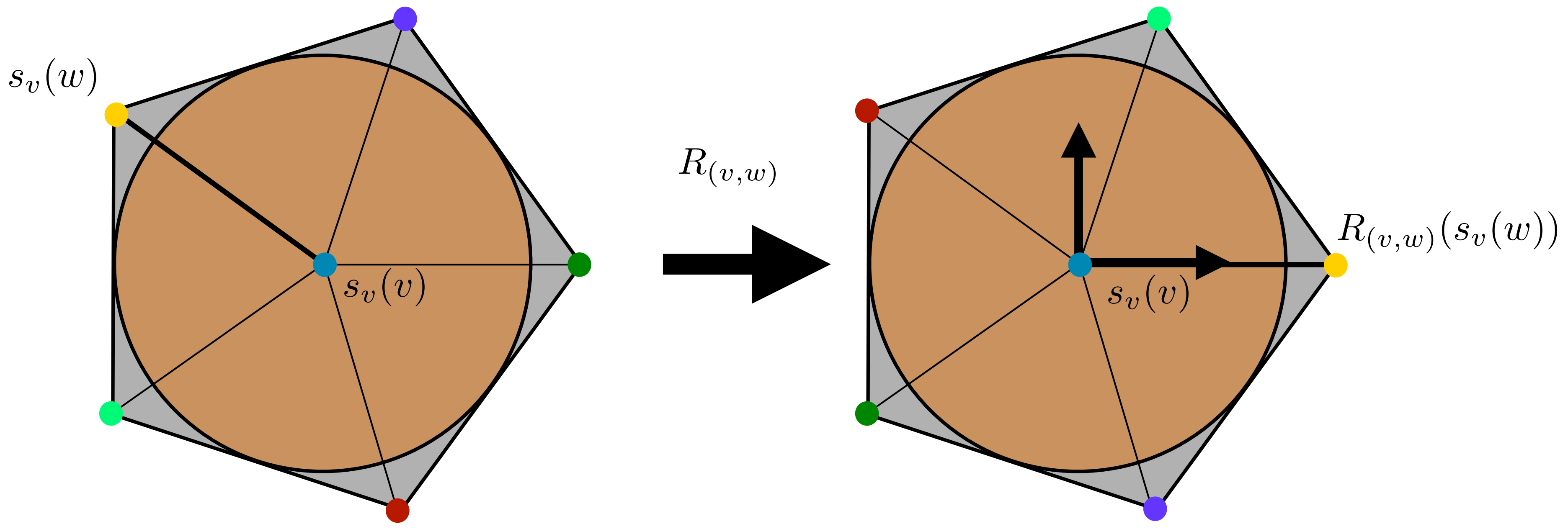
Why do we need this attribute?

To find the rotation angle of $R_{(v,w)}$ (or of its inverse), where $[v, w]$ is the edge associated with h and v is the origin vertex of h.

Data Structure

Why do we need this attribute?

To find the rotation angle of $R_{(v,w)}$ (or of its inverse), where $[v,w]$ is the edge associated with h and v is the origin vertex of h.



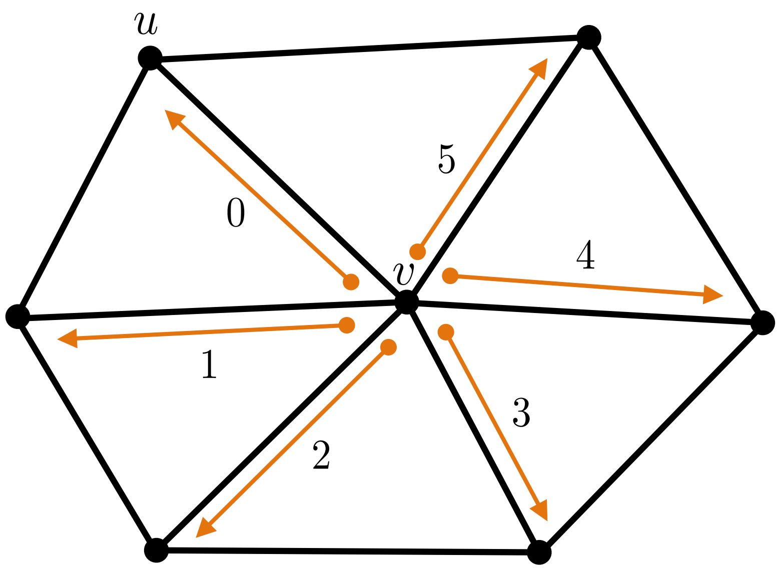
Data Structure

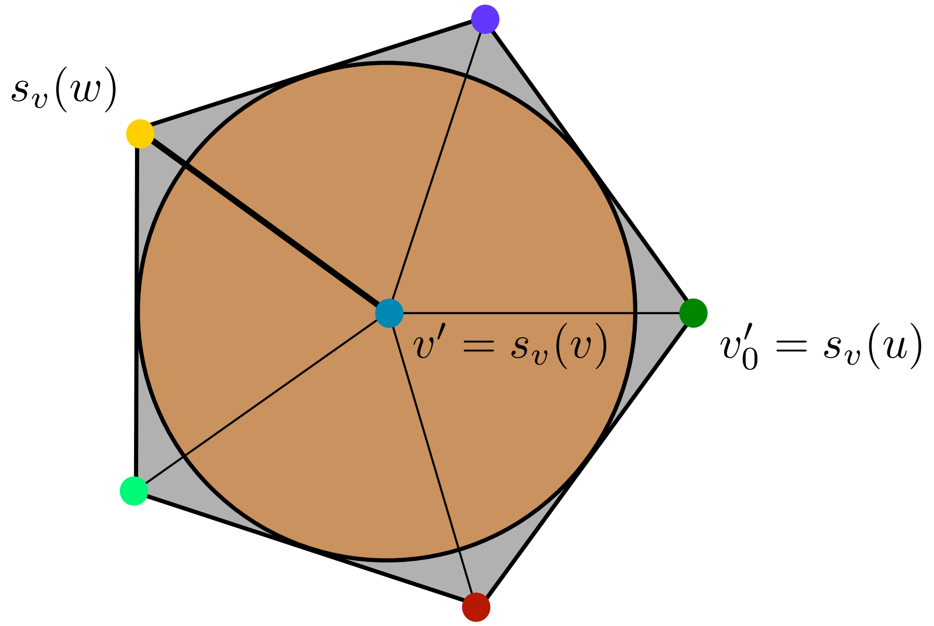
Data Structure

First, we assume that the half-edge with *id* equals to 0 is the one whose associated edge is mapped to the edge of T_v with endpoints $(2 \cdot l(v), 0)$ and $(2 \cdot l(v) + 1, 0)$ (i.e., the vertices v' and v'_0).

Data Structure

First, we assume that the half-edge with *id* equals to 0 is the one whose associated edge is mapped to the edge of T_v with endpoints $(2 \cdot l(v), 0)$ and $(2 \cdot l(v) + 1, 0)$ (i.e., the vertices v' and v'_0).





Data Structure

Data Structure

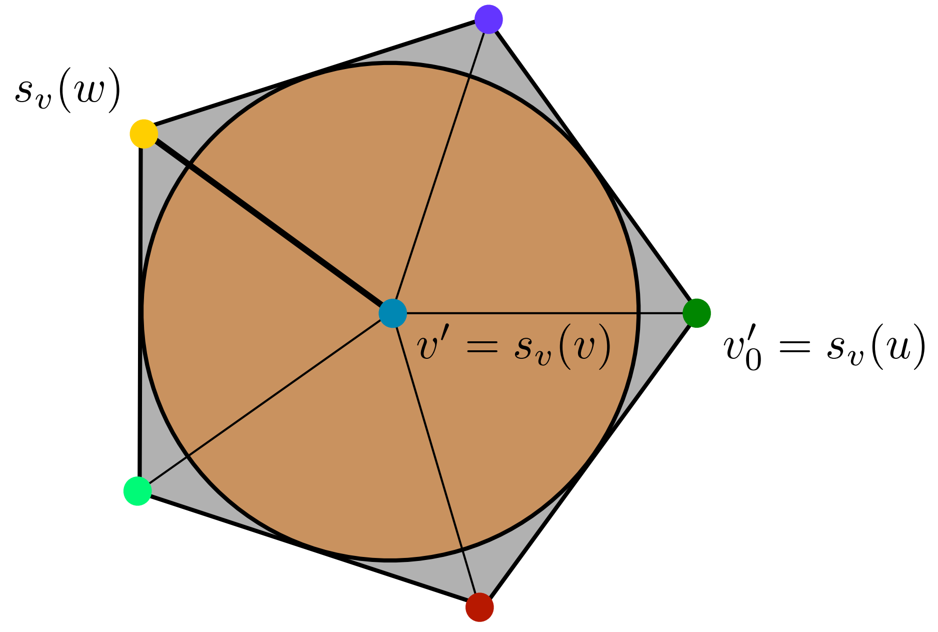
So, the rotation angle is equal to

$$-\frac{2\pi \cdot \mathit{id}}{m_v}\,.$$

Data Structure

So, the rotation angle is equal to

$$-\frac{2\pi \cdot \mathit{id}}{m_v}\,.$$



Data Structure

Data Structure

Edges and Triangles have no attributes related to the construction.

Data Structure

Edges and Triangles have no attributes related to the construction.

When using our construction with PN triangle surfaces, the PN triangle control points of each patch become triangle attributes.

Data Structure

Edges and Triangles have no attributes related to the construction.

When using our construction with PN triangle surfaces, the PN triangle control points of each patch become triangle attributes.

Likewise, when using subdivision surfaces, some parameters used by Stam's exact evaluation algorithm become triangle attributes.

Data Structure

Data Structure

At this point, you may wonder where we store P-polygons, p-domains, gluing domains, and triangulations associated with P-polygons.

Data Structure

At this point, you may wonder where we store P-polygons, p-domains, gluing domains, and triangulations associated with P-polygons.

WE DON'T!

Data Structure

At this point, you may wonder where we store P-polygons, p-domains, gluing domains, and triangulations associated with P-polygons.

WE DON'T!

Those are abstractions we need to describe the construction only.

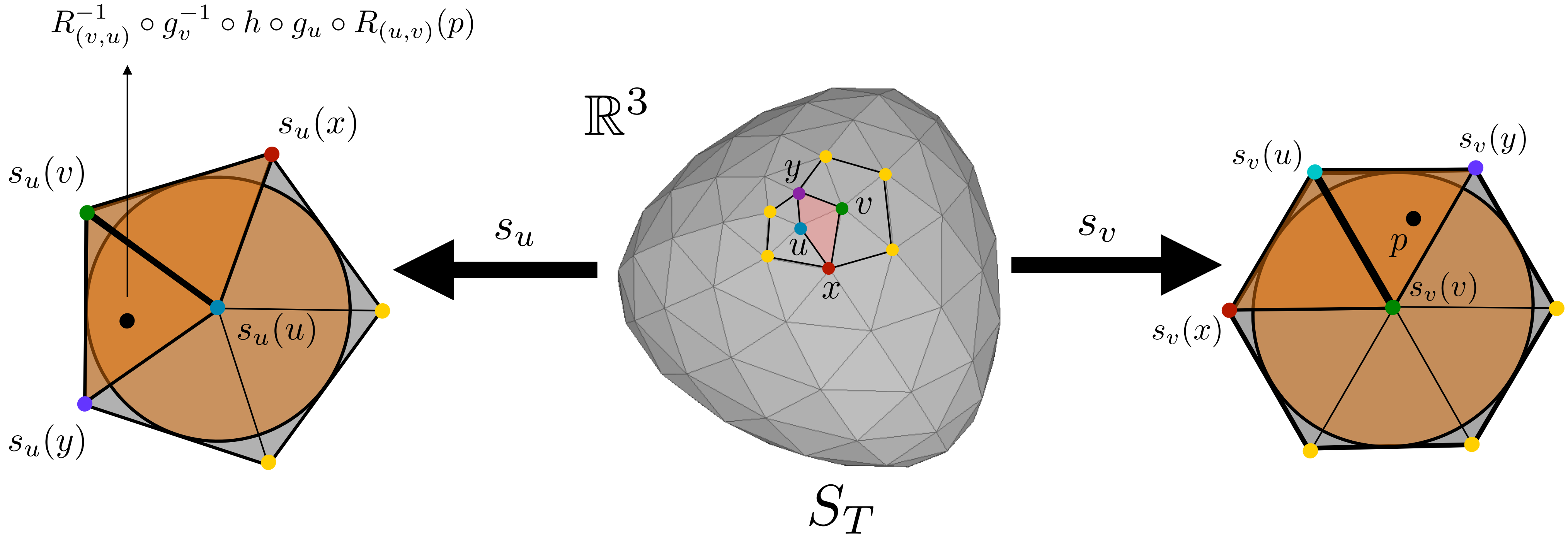
Algorithms

Algorithms

How can we compute φ_{uv} at some $p \in \Omega_v$?

Algorithms

How can we compute φ_{uv} at some $p \in \Omega_v$?



Algorithms

Algorithms

To start with, let us assume that we have a pointer to the half-edge, h_{vu}, with origin at v and associated with the edge $[u, v]$.

Algorithms

To start with, let us assume that we have a pointer to the half-edge, h_{vu}, with origin at v and associated with the edge $[u, v]$.

Recall that

$$\varphi_{uv}(p) = R^{-1}_{(u,v)} \circ g_u^{-1} \circ h \circ g_v \circ R_{(v,u)}(p)\,.$$

Algorithms

Algorithms

Since we have a pointer to h_{vu}, we can recover its *id* number and then compute the rotation angle of $R_{(v,u)}$, as we saw before.

Algorithms

Since we have a pointer to h_{vu}, we can recover its *id* number and then compute the rotation angle of $R_{(v,u)}$, as we saw before.

Using `MATE` we can get h_{uv}, recover its *id* number and then compute the rotation angle of $R^{-1}_{(u,v)}$, which is -1 times the angle of $R_{(u,v)}$.

Algorithms

Since we have a pointer to h_{vu}, we can recover its *id* number and then compute the rotation angle of $R_{(v,u)}$, as we saw before.

Using `MATE` we can get h_{uv}, recover its *id* number and then compute the rotation angle of $R^{-1}_{(u,v)}$, which is -1 times the angle of $R_{(u,v)}$.

So, we are left with g_v, g_u^{-1}, and h, but h is straightforward.

Algorithms

Algorithms

Recall that

$$g_v = \Pi^{-1} \circ f_v \circ \Pi \circ t_v \,,$$

where t_v is the translation that takes $(2 \cdot l(v), 0)$ to $(0, 0)$, Π (resp. Π^{-1}) is the function that converts Cartesian (resp. polar) into polar (Cartesian) coordinates, and f_v is given by

$$f_v(q) = f_v((\theta, r)) = \left(\frac{m_v}{6} \cdot \theta, \frac{\cos(\pi/6)}{\cos(\pi/m_v)} \cdot r \right) \,,$$

where (θ, r) are the polar coordinates of $q \in (-\pi, \pi] \times \mathbb{R}_+$.

Algorithms

Algorithms

In practice, we assume that every p-domain is centered at $(0,0)$, which makes $g_v = \Pi^{-1} \circ f_v \circ \Pi$. Fine for the sake of implementation!

Algorithms

In practice, we assume that every p-domain is centered at $(0, 0)$, which makes $g_v = \Pi^{-1} \circ f_v \circ \Pi$. Fine for the sake of implementation!

Since we have a pointer to h_{vu}, we can access a pointer to v, and thus we can get m_v. Along with the polar coordinates of q (which are immediate to compute), this is all we need to compute g_v.

Algorithms

In practice, we assume that every p-domain is centered at $(0,0)$, which makes $g_v = \Pi^{-1} \circ f_v \circ \Pi$. Fine for the sake of implementation!

Since we have a pointer to h_{vu}, we can access a pointer to v, and thus we can get m_v. Along with the polar coordinates of q (which are immediate to compute), this is all we need to compute g_v.

Likewise, since we have a pointer to h_{uv}, we can access a pointer to u, and thus we can can get m_u. That's all we need to compute g_u^{-1}.

Algorithms

Algorithms

Let us turn our attention to the computation of a point on the PPS.

Algorithms

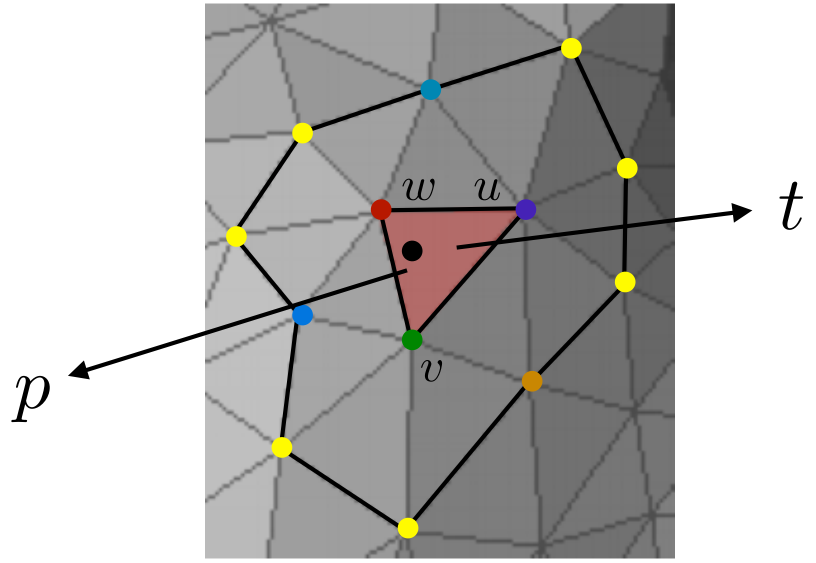
Let us turn our attention to the computation of a point on the PPS.

As we have said before, we assume that we are given barycentric coordinates of a point p in a triangle $t = [v, u, w]$ of S_T.

Algorithms

Let us turn our attention to the computation of a point on the PPS.

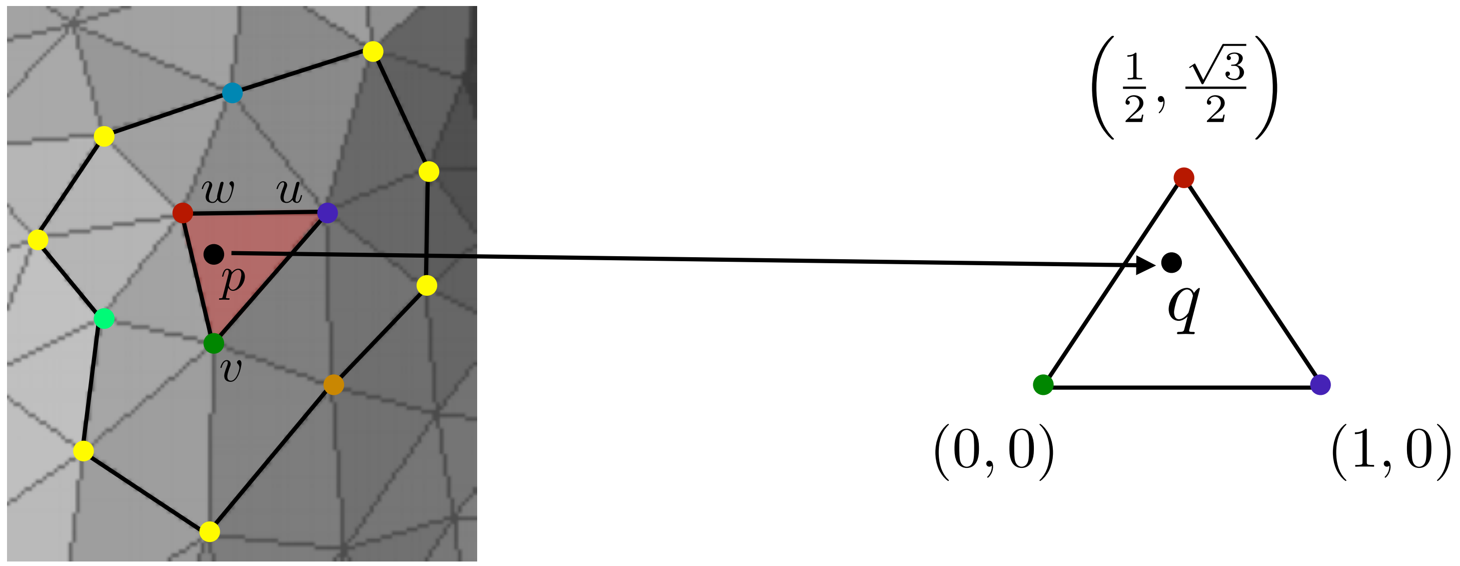
As we have said before, we assume that we are given barycentric coordinates of a point p in a triangle $t = [v, u, w]$ of S_T.



Algorithms

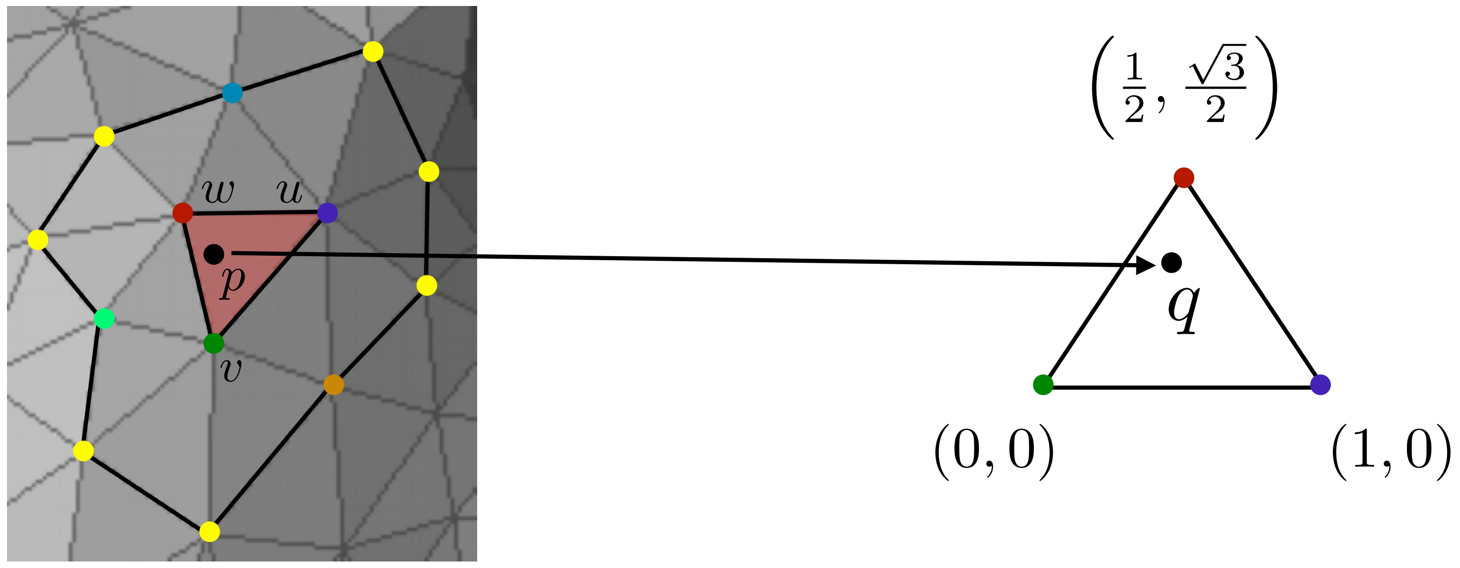
Algorithms

Map p to an equilateral triangle in $\mathbb{R}^2$.



Algorithms

Map p to an equilateral triangle in $\mathbb{R}^2$.



We can do that by using barycentric coordinates.

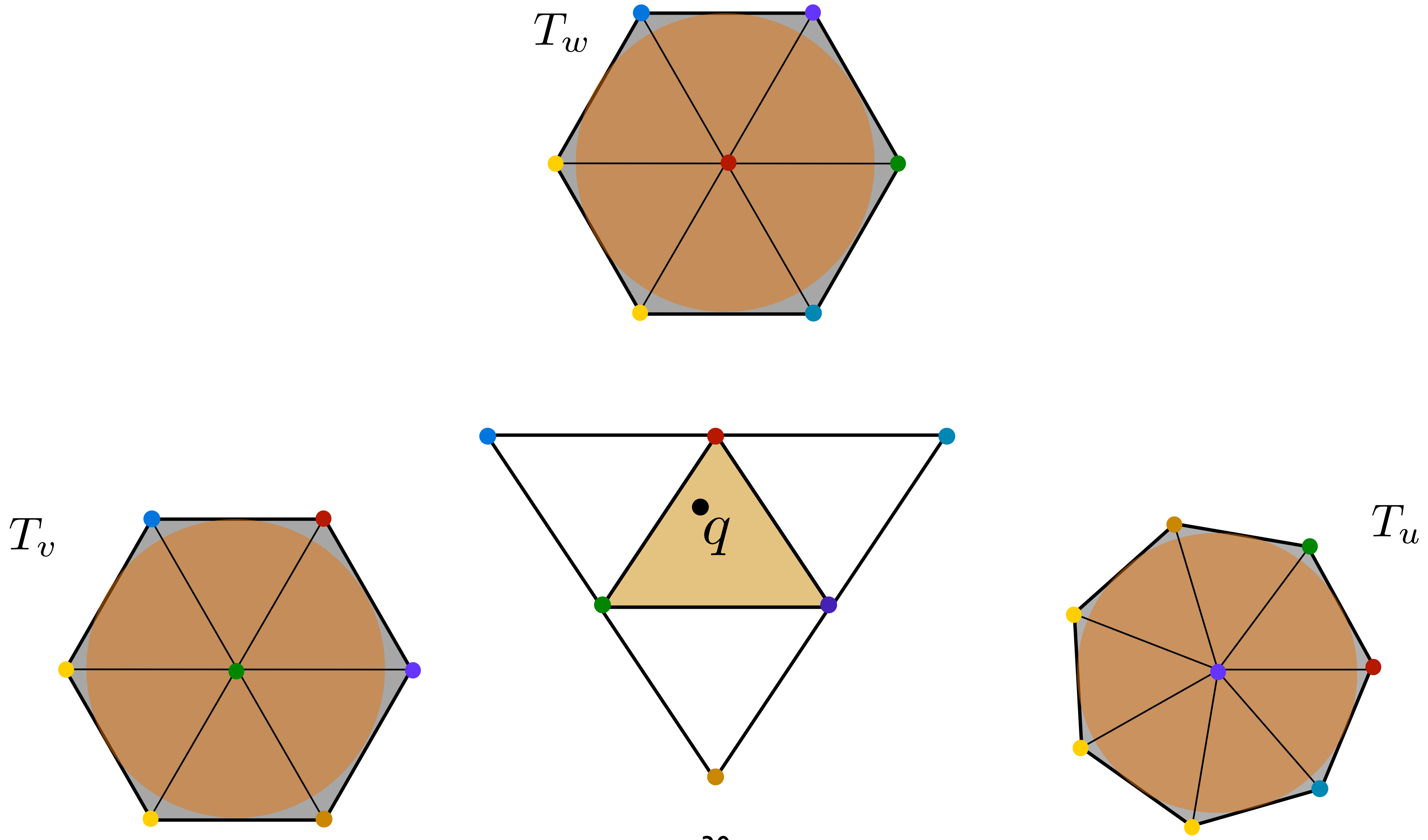
Algorithms

Algorithms

Finally, we map q to one of the p-domains: Ω_v, Ω_u, and Ω_w.

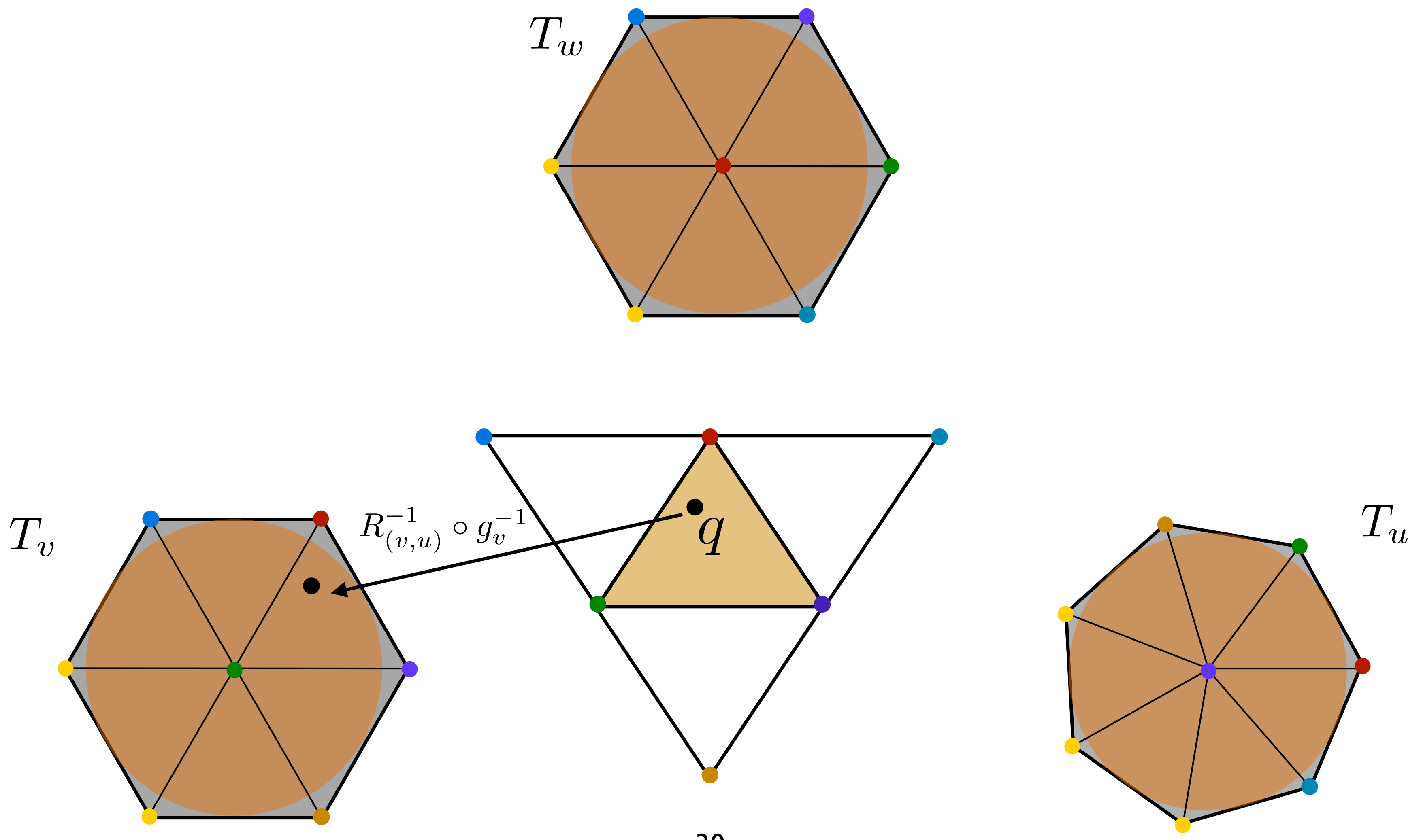
Algorithms

Finally, we map q to one of the p-domains: Ω_v, Ω_u, and Ω_w.



Algorithms

Finally, we map q to one of the p-domains: Ω_v, Ω_u, and Ω_w.



Algorithms

Algorithms

It might very well be the case that $x = R^{-1}_{(v,u)} \circ g_v^{-1}(q)$ is not in Ω_v.

Algorithms

It might very well be the case that $x = R^{-1}_{(v,u)} \circ g_v^{-1}(q)$ is not in Ω_v.

If that's the case, we choose either u or w and repeat the process.

Algorithms

It might very well be the case that $x = R^{-1}_{(v,u)} \circ g_v^{-1}(q)$ is not in Ω_v.

If that's the case, we choose either u or w and repeat the process.

By construction, we are guaranteed to succeed with one of them!

Algorithms

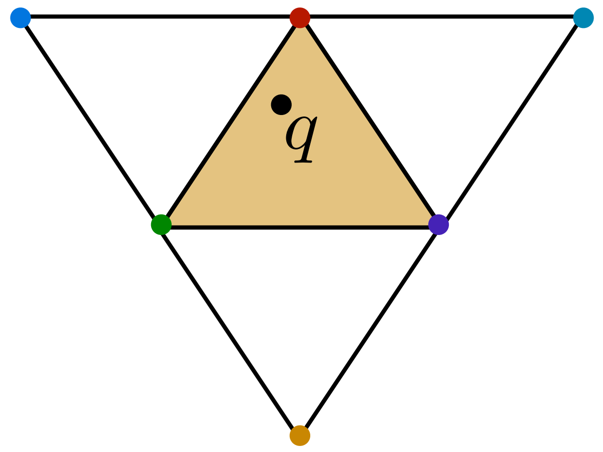
Algorithms

BE CAREFUL:

Algorithms

BE CAREFUL:

Every time we try a new vertex, the barycentric coordinates of q must change, as the chosen vertex is put in correspondence with the vertex $(0, 0)$ of the canonical triangle (the one colored with "green").



Algorithms

Algorithms

To compute g_v^{-1} we need m_v, the degree of v and the polar coordinates of q.

Algorithms

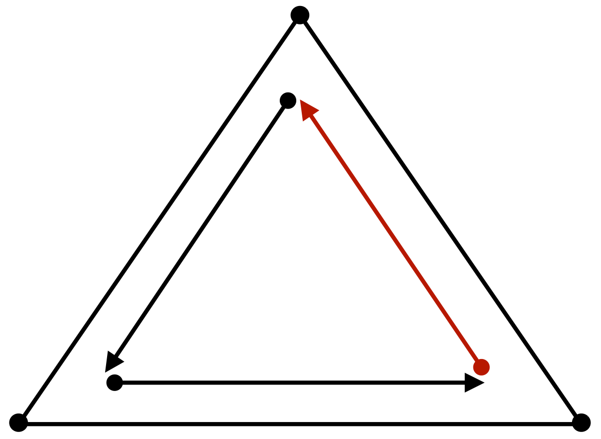
To compute g_v^{-1} we need m_v, the degree of v and the polar coordinates of q.

The parameter m_v can be easily obtained from a pointer to the half-edge h_{vu} of the given triangle t that contains the point p in S_T.

Algorithms

To compute g_v^{-1} we need m_v, the degree of v and the polar coordinates of q.

The parameter m_v can be easily obtained from a pointer to the half-edge h_{vu} of the given triangle t that contains the point p in S_T.



Algorithms

Algorithms

Let

$$x = R_{(v,u)}^{-1} \circ g_v^{-1}(q), \quad y = \varphi_{uv}(x), \quad z = \varphi_{wv}(x)\,.$$

Algorithms

Let

$$x = R_{(v,u)}^{-1} \circ g_v^{-1}(q), \quad y = \varphi_{uv}(x), \quad z = \varphi_{wv}(x)\,.$$

To compute φ_{uv} and φ_{wv} we need m_u and m_w, which can also be obtained from pointers to the half-edges h_{uw} and h_{wv}, which in turn can be obtained from a pointer to the given triangle t.

Algorithms

Let

$$x = R^{-1}_{(v,u)} \circ g_v^{-1}(q), \quad y = \varphi_{uv}(x), \quad z = \varphi_{wv}(x)\,.$$

To compute φ_{uv} and φ_{wv} we need m_u and m_w, which can also be obtained from pointers to the half-edges h_{uw} and h_{wv}, which in turn can be obtained from a pointer to the given triangle t.

We also need m_v and the half-edge *id* numbers (for the rotation functions), but we can get all that through the half-edge pointers.

Algorithms

Algorithms

Compute

$$\gamma_v(x), \quad \gamma_u(y) \quad \text{and} \quad \gamma_w(z)\,.$$

Algorithms

Compute

$$\gamma_v(x), \quad \gamma_u(y) \quad \text{and} \quad \gamma_w(z)\,.$$

Recall that

$$\gamma_v(x) = \xi(\|p - (2 \cdot l(v), 0)\|) = \xi(\|p - (0,0)\|)\,,$$

as we consider $l(v) = 0$ (for the sake of implementation ONLY),

Algorithms

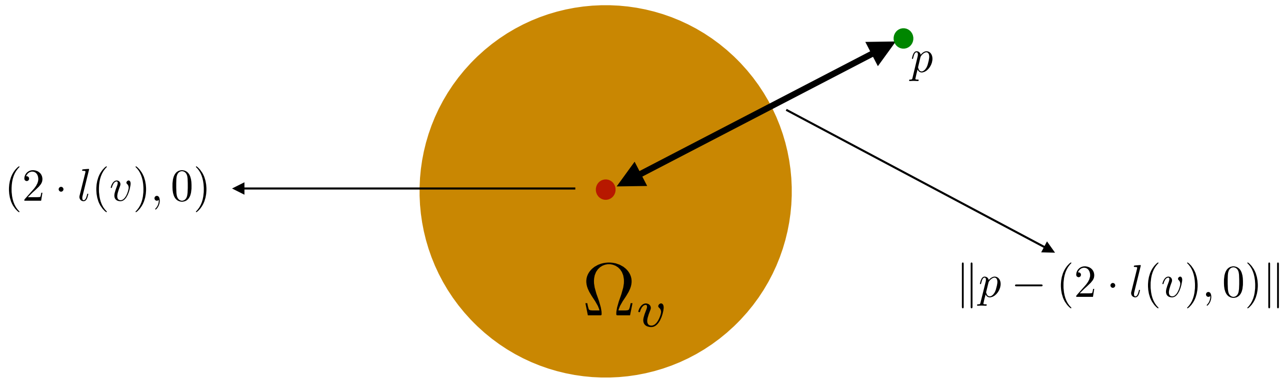
Compute

$$\gamma_v(x), \quad \gamma_u(y) \quad \text{and} \quad \gamma_w(z)\,.$$

Recall that

$$\gamma_v(x) = \xi(\|p - (2\cdot l(v), 0)\|) = \xi(\|p - (0,0)\|)\,,$$

as we consider $l(v) = 0$ (for the sake of implementation ONLY),



Algorithms

Algorithms

$\xi : \mathbb{R} \to \mathbb{R}$ is such that, for every $t \in \mathbb{R}$,

$$\xi(t) = \begin{cases} 1 & \text{if } t \leq L_1 \\ 0 & \text{if } t \geq L_2 \\ h(L)/(h(L) + h(1 - L)) & \text{otherwise}\,, \end{cases}$$

where

$$\xi(t) = \begin{cases} 1 & \text{if } t \leq 0 \\ 0 & \text{if } t \geq 1 \\ e^{\frac{2 \cdot e^{-1/t}}{t-1}} & \text{otherwise}\,, \end{cases}$$

L_1, L_2 are constants, with $0 < L_1 < L_2 < 1$, and $L = (t - L_1)/(L_2 - L_1)$.

Algorithms

Algorithms

Since we let $L_1 = 0.25 \cdot L_2$ and $L_2 = \cos(\pi/m_v)$, we need m_v, which can be obtained through a pointer to v, but such a pointer can be accessed by a pointer to h_{vu} (which we have).

Algorithms

Since we let $L_1 = 0.25 \cdot L_2$ and $L_2 = \cos(\pi/m_v)$, we need m_v, which can be obtained through a pointer to v, but such a pointer can be accessed by a pointer to h_{vu} (which we have).

For γ_u and γ_w, we need m_u and m_w, which we can get in a similar way.

Algorithms

Since we let $L_1 = 0.25 \cdot L_2$ and $L_2 = \cos(\pi/m_v)$, we need m_v, which can be obtained through a pointer to v, but such a pointer can be accessed by a pointer to h_{vu} (which we have).

For γ_u and γ_w, we need m_u and m_w, which we can get in a similar way.

Let $W_v = \gamma_v(x)$, $W_u = \gamma_u(y)$, and $W_w = \gamma_w(z)$.

Algorithms

Algorithms

If $W_v \neq 0$, compute $\psi_v(x)$ and let $S_v = \psi_v(x) \cdot W_v$. Otherwise, let $S_v = 0$.

Algorithms

If $W_v \neq 0$, compute $\psi_v(x)$ and let $S_v = \psi_v(x) \cdot W_v$. Otherwise, let $S_v = 0$.

If $W_u \neq 0$, compute $\psi_u(y)$ and let $S_u = \psi_u(y) \cdot W_u$. Otherwise, let $S_u = 0$.

If $W_w \neq 0$, compute $\psi_w(z)$ and let $S_w = \psi_w(z) \cdot W_w$. Otherwise, let $S_w = 0$.

Algorithms

If $W_v \neq 0$, compute $\psi_v(x)$ and let $S_v = \psi_v(x) \cdot W_v$. Otherwise, let $S_v = 0$.

If $W_u \neq 0$, compute $\psi_u(y)$ and let $S_u = \psi_u(y) \cdot W_u$. Otherwise, let $S_u = 0$.

If $W_w \neq 0$, compute $\psi_w(z)$ and let $S_w = \psi_w(z) \cdot W_w$. Otherwise, let $S_w = 0$.

It is true that $W_v + W_u + W_w \neq 0$.

Algorithms

Algorithms

Finally, the point on the PPS is given by

$$\frac{S_v + S_u + S_w}{W_v + W_u + W_w}\,.$$

Algorithms

Finally, the point on the PPS is given by

$$\frac{S_v + S_u + S_w}{W_v + W_u + W_w}\,.$$

What we just did was to compute

$$\theta_v(x) = \theta_u(y) = \theta_w(z)\,.$$

Conclusions

Conclusions

- Currently, we only have the point evaluation procedure in our API.

Conclusions

- Currently, we only have the point evaluation procedure in our API.
- To make the PPS useful for practical applications, this API must be extended to include several fundamental operations, such as computation of derivatives, ray intersection, etc.

Suggested Reading

Suggested Reading

Unfortunately, there is no proper documentation of the code other than the inline comments. This must change in the future.

Acknowledgments

Acknowledgments

We've worked with the following people to obtain the results shown here:

Acknowledgments

We've worked with the following people to obtain the results shown here:

Dianna Xu (Brynmawr College)

Dimas Martínez Morera (UFAL)

Jean Gallier (UPenn)

Luis Gustavo Nonato (ICMC-USP)

Thank You!