# A New Algorithm for Generating Quadrilateral Meshes and Its Application to FE-Based Image Registration


Suneeta Ramaswami
Rutgers University, USA
*rsuneeta@camden.rutgers.edu*

Marcelo Siqueira
University of Pennsylvania, USA
*marcelos@seas.upenn.edu*

Tessa Sundaram
University of Pennsylvania, USA
*tessa@mail.med.upenn.edu*

Jean Gallier
University of Pennsylvania, USA
*jean@cis.upenn.edu*

James Gee
University of Pennsylvania, USA
*gee@rad.upenn.edu*

# Introduction

In this talk we present

# Introduction

In this talk we present

- a new algorithm to generate *strictly* convex quadrangulations of provably small size of polygonal regions with or without polygonal holes,

# Introduction

In this talk we present

- a new algorithm to generate $strictly$ convex quadrangulations of provably small size of polygonal regions with or without polygonal holes,

- an approach to create quadrilateral meshes from 2D images of the human brain using our algorithm,

# Introduction

In this talk we present

- a new algorithm to generate *strictly* convex quadrangulations of provably small size of polygonal regions with or without polygonal holes,
- an approach to create quadrilateral meshes from 2D images of the human brain using our algorithm,
- and a comparison of the performance of a FE-based image registration method with respect to distinct input image meshes, including the ones generated by our algorithm.

# The Algorithm

## Problem

- Given a bounded polygonal region $\mathcal{P}$ with $n$ vertices and $h \geq 0$ polygonal holes, obtain a strictly convex quadrangulation of $\mathcal{P}$.

# The Algorithm

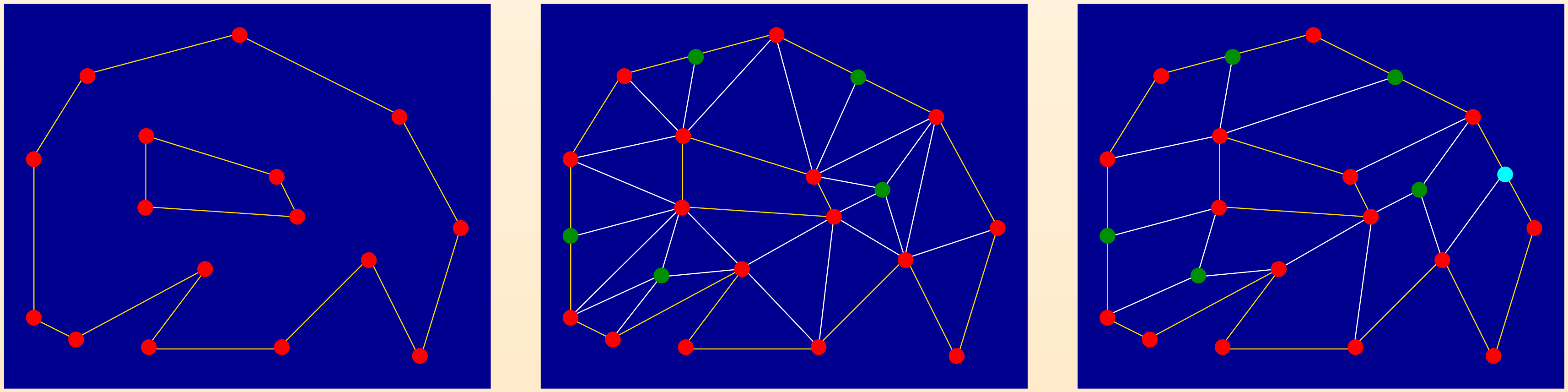
## Problem

- Given a bounded polygonal region $\mathcal{P}$ with $n$ vertices and $h \geq 0$ polygonal holes, obtain a strictly convex quadrangulation of $\mathcal{P}$.

## Conversion from a Triangulation

- Obtain a triangulation $\mathcal{T}$ of $\mathcal{P}$ such that $V_{\mathcal{P}} \subseteq V_{\mathcal{T}}$ and $|\mathcal{T}| = \mathcal{P}$.

- Convert $\mathcal{T}$ into a quadrangulation $\mathcal{Q}$ of $\mathcal{P}$.



# The Algorithm

## Our Solution

- We propose an algorithm for the conversion step: Given a triangulation $\mathcal{T}$ of $\mathcal{P}$, obtain a quadrangulation $\mathcal{Q}$ of $\mathcal{P}$.

# The Algorithm

## Our Solution

- We propose an algorithm for the conversion step: Given a triangulation $\mathcal{T}$ of $\mathcal{P}$, obtain a quadrangulation $\mathcal{Q}$ of $\mathcal{P}$.
- Recall that if $V_{\mathcal{T}}$ has $m$ vertices then the number $t$ of triangles of $\mathcal{T}$ is $t = 2m + 2h - 2 - m_b$, where $m_b \geq n$ is the number of vertices on the boundary of $\mathcal{T}$.

# The Algorithm

## Our Solution

- We propose an algorithm for the conversion step: Given a triangulation $\mathcal{T}$ of $\mathcal{P}$, obtain a quadrangulation $\mathcal{Q}$ of $\mathcal{P}$.

- Recall that if $V_{\mathcal{T}}$ has $m$ vertices then the number $t$ of triangles of $\mathcal{T}$ is $t = 2m + 2h - 2 - m_b$, where $m_b \geq n$ is the number of vertices on the boundary of $\mathcal{T}$.

- The quadrangulation $\mathcal{Q}$ generated by our algorithm contains $V_{\mathcal{T}}$ and at most $t + 2$ more vertices (Steiner points).

# The Algorithm

## Our Solution

- We propose an algorithm for the conversion step: Given a triangulation $\mathcal{T}$ of $\mathcal{P}$, obtain a quadrangulation $\mathcal{Q}$ of $\mathcal{P}$.

- Recall that if $V_{\mathcal{T}}$ has $m$ vertices then the number $t$ of triangles of $\mathcal{T}$ is $t = 2m + 2h - 2 - m_b$, where $m_b \geq n$ is the number of vertices on the boundary of $\mathcal{T}$.

- The quadrangulation $\mathcal{Q}$ generated by our algorithm contains $V_{\mathcal{T}}$ and at most $t + 2$ more vertices (Steiner points).

- The quadrangulation $\mathcal{Q}$ generated by our algorithm contains at most $\frac{3}{2}t$ quadrilaterals.

# The Algorithm

## Our Solution

- We propose an algorithm for the conversion step: Given a triangulation $\mathcal{T}$ of $\mathcal{P}$, obtain a quadrangulation $\mathcal{Q}$ of $\mathcal{P}$.
- Recall that if $V_{\mathcal{T}}$ has $m$ vertices then the number $t$ of triangles of $\mathcal{T}$ is $t = 2m + 2h - 2 - m_b$, where $m_b \geq n$ is the number of vertices on the boundary of $\mathcal{T}$.
- The quadrangulation $\mathcal{Q}$ generated by our algorithm contains $V_{\mathcal{T}}$ and at most $t+2$ more vertices (Steiner points).
- The quadrangulation $\mathcal{Q}$ generated by our algorithm contains at most $\frac{3}{2}t$ quadrilaterals.
- The size and time complexity of our algorithm are $\mathcal{O}(t)$.

# The Algorithm

## Our Solution

- We propose an algorithm for the conversion step: Given a triangulation $\mathcal{T}$ of $\mathcal{P}$, obtain a quadrangulation $\mathcal{Q}$ of $\mathcal{P}$.

- Recall that if $V_{\mathcal{T}}$ has $m$ vertices then the number $t$ of triangles of $\mathcal{T}$ is $t = 2m + 2h - 2 - m_b$, where $m_b \geq n$ is the number of vertices on the boundary of $\mathcal{T}$.

- The quadrangulation $\mathcal{Q}$ generated by our algorithm contains $V_{\mathcal{T}}$ and at most $t + 2$ more vertices (Steiner points).

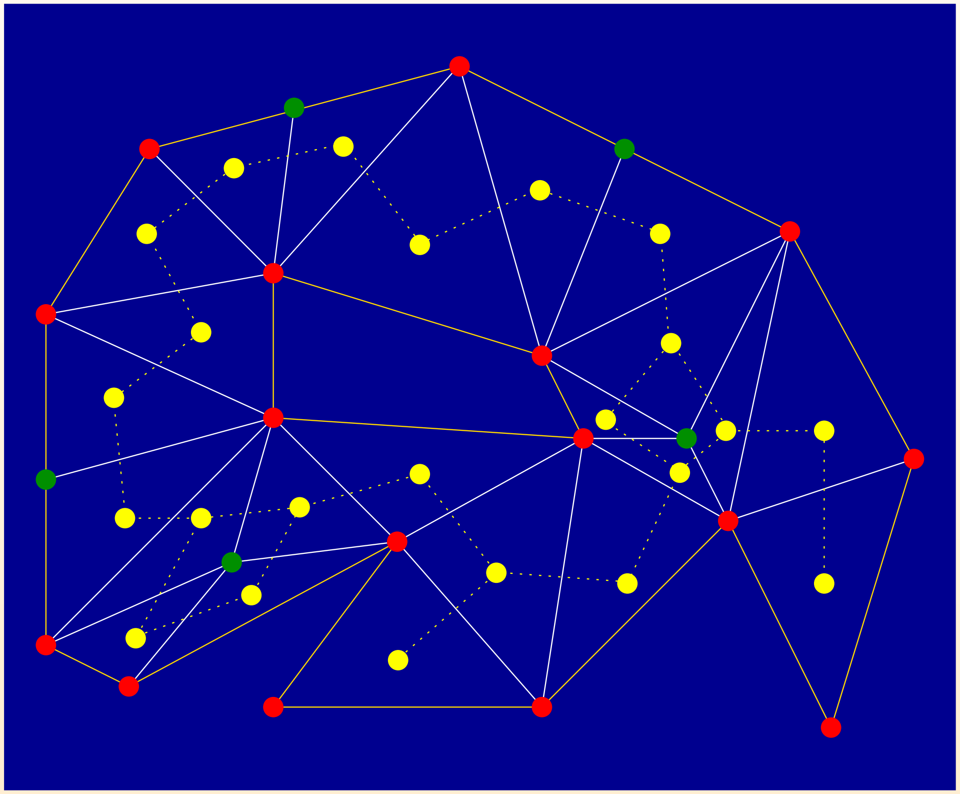
- The quadrangulation $\mathcal{Q}$ generated by our algorithm contains at most $\frac{3}{2}t$ quadrilaterals.

- The size and time complexity of our algorithm are $\mathcal{O}(t)$.

- The above bounds are better than the ones provided by previous indirect algorithms that also provide theoretical bounds on the size of the output quadrangulation.

# The Algorithm

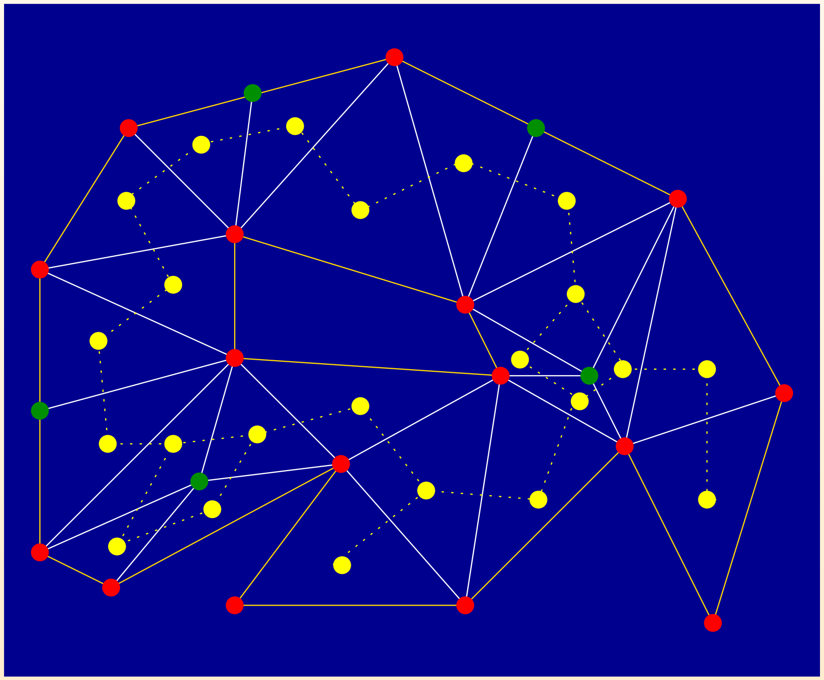
- Build the dual graph $G$ of $\mathcal{T}$.

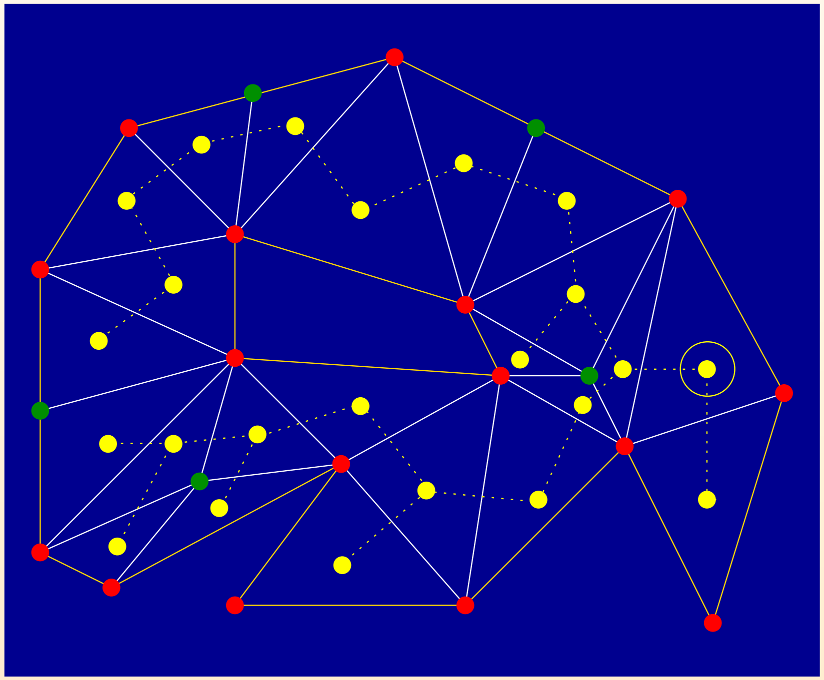


# The Algorithm

- Build a *rooted* spanning tree $T_G$ of $G$ such that the root of $T_G$ is a vertex corresponding to a triangle of $\mathcal{T}$ incident to an edge of the boundary of $\mathcal{T}$.

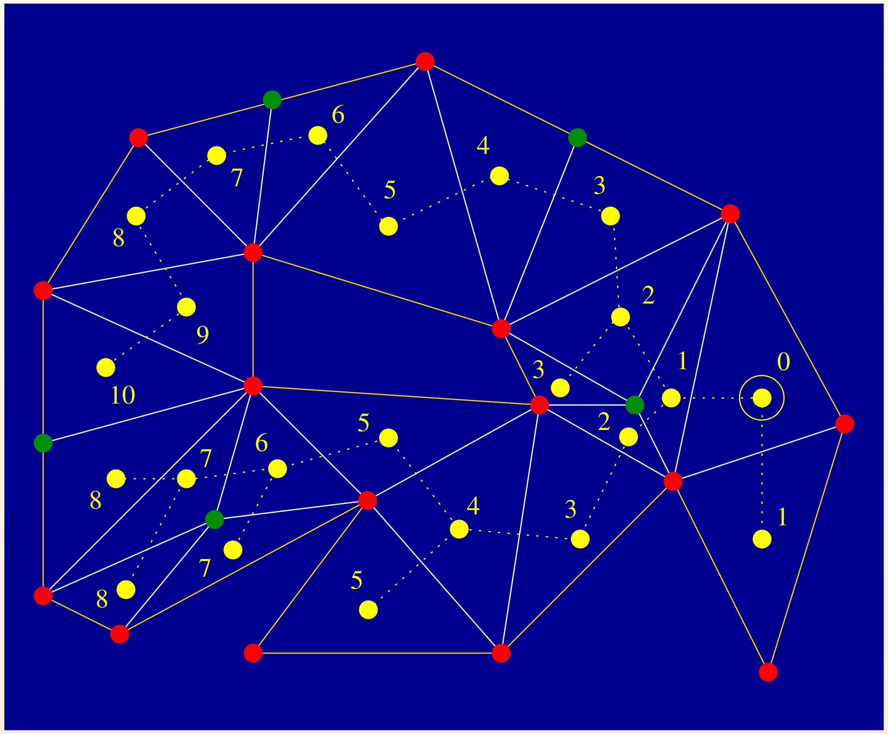
- We carry out a Breadth-First Search (BFS) on $G$ to obtain $T_G$.

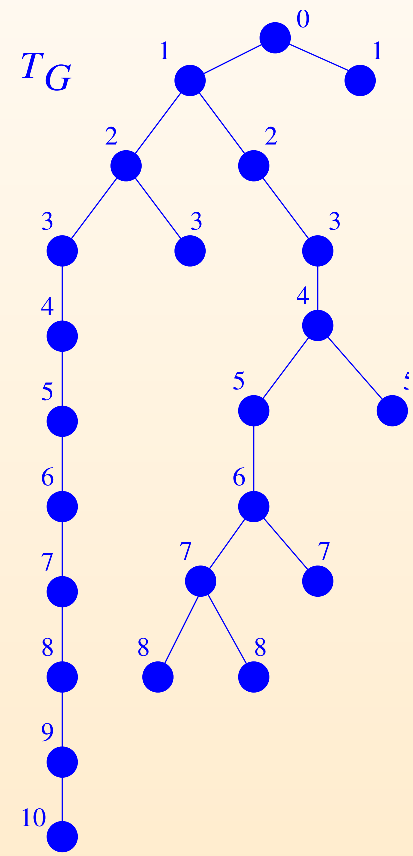




# The Algorithm

- Let $k$ be the deepest level of $T_G$, and let $V_i$ $(0 \leq i \leq k)$ be the set of vertices of $T_G$ at level $i$.





# The Algorithm

- Our algorithm converts $\mathcal{T}$ into a quadrangulation $\mathcal{Q}$ of $|\mathcal{T}|$ by processing the sets $V_k, V_{k-1}, \ldots, V_1, V_0$ one at a time and in this order. That is, the algorithm traverses $T_G$ per level in a bottom-up fashion.

# The Algorithm

- Our algorithm converts $\mathcal{T}$ into a quadrangulation $\mathcal{Q}$ of $|\mathcal{T}|$ by processing the sets $V_k, V_{k-1}, \ldots, V_1, V_0$ one at a time and in this order. That is, the algorithm traverses $T_G$ per level in a bottom-up fashion.

- After processing a vertex $v$ in $V_i$ $(0 \leq i \leq k)$, the algorithm removes $v$ from both $T_G$ and $V_i$.

# The Algorithm

- Our algorithm converts $\mathcal{T}$ into a quadrangulation $\mathcal{Q}$ of $|\mathcal{T}|$ by processing the sets $V_k, V_{k-1}, \ldots, V_1, V_0$ one at a time and in this order. That is, the algorithm traverses $T_G$ per level in a bottom-up fashion.

- After processing a vertex $v$ in $V_i$ $(0 \leq i \leq k)$, the algorithm removes $v$ from both $T_G$ and $V_i$.

- From the two statements above, a vertex $v$ in $V_i$ $(0 \leq i \leq k)$ must be a leaf by the time it is processed by the algorithm.

# The Algorithm

- Our algorithm converts $\mathcal{T}$ into a quadrangulation $\mathcal{Q}$ of $|\mathcal{T}|$ by processing the sets $V_k, V_{k-1}, \ldots, V_1, V_0$ one at a time and in this order. That is, the algorithm traverses $T_G$ per level in a bottom-up fashion.

- After processing a vertex $v$ in $V_i$ $(0 \leq i \leq k)$, the algorithm removes $v$ from both $T_G$ and $V_i$.

- From the two statements above, a vertex $v$ in $V_i$ $(0 \leq i \leq k)$ must be a leaf by the time it is processed by the algorithm.

- For any vertex $v \in T_G$, let $par(v)$ denote the parent of $v$ in $T_G$. When processing a vertex $v$ in $V_i$ $(0 \leq i \leq k)$, the algorithm considers either the vertex $v$ itself, or the vertices in the subtree of $T_G$ rooted at $par(v)$, or the vertices in the subtree of $T_G$ rooted at $par(par(v))$.

# The Algorithm

- Our algorithm converts $\mathcal{T}$ into a quadrangulation $\mathcal{Q}$ of $|\mathcal{T}|$ by processing the sets $V_k, V_{k-1}, \ldots, V_1, V_0$ one at a time and in this order. That is, the algorithm traverses $T_G$ per level in a bottom-up fashion.

- After processing a vertex $v$ in $V_i$ $(0 \leq i \leq k)$, the algorithm removes $v$ from both $T_G$ and $V_i$.

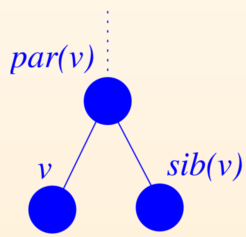
- From the two statements above, a vertex $v$ in $V_i$ $(0 \leq i \leq k)$ must be a leaf by the time it is processed by the algorithm.

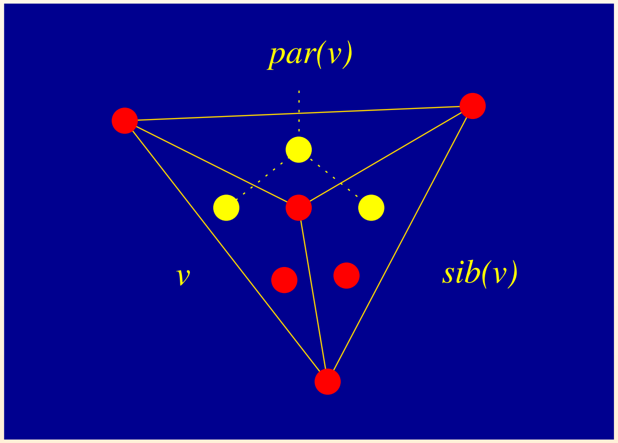
- For any vertex $v \in T_G$, let $par(v)$ denote the parent of $v$ in $T_G$. When processing a vertex $v$ in $V_i$ $(0 \leq i \leq k)$, the algorithm considers either the vertex $v$ itself, or the vertices in the subtree of $T_G$ rooted at $par(v)$, or the vertices in the subtree of $T_G$ rooted at $par(par(v))$.

- Initially, every vertex $v$ in $T_G$ corresponds to one triangle in $\mathcal{T}$, but as the algorithm starts traversing and pruning $T_G$, a vertex $v$ in $T_G$ can correspond to either a triangle in $\mathcal{T}$, a *non-empty triangle*, a *degenerate quadrilateral*, or a *degenerate pentagon*.

# The Algorithm

- What is a non-empty triangle? How does it show up?

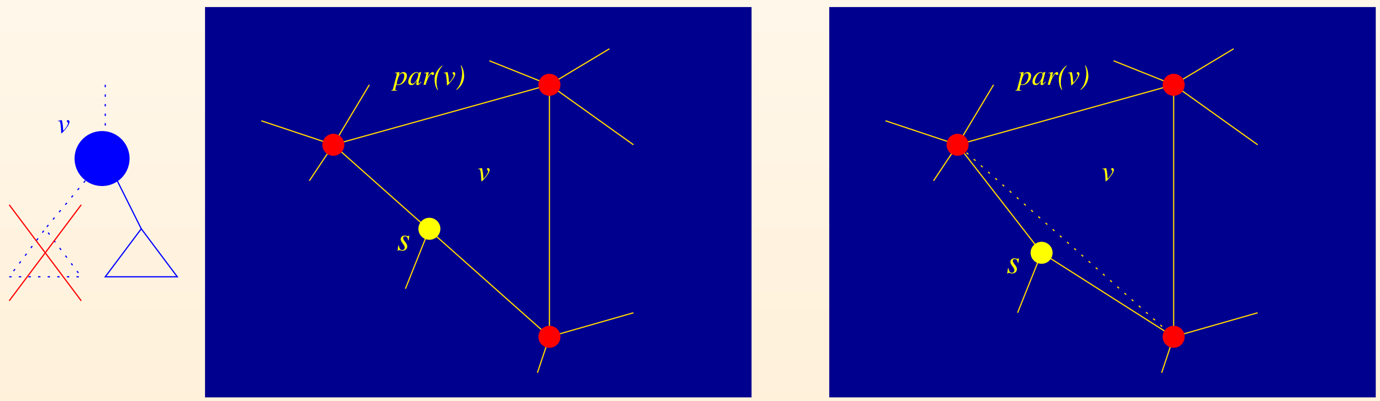



- Note that if $v \in T_G$ corresponds to a non-empty triangle, then $v$ is a leaf of $T_G$.

# The Algorithm

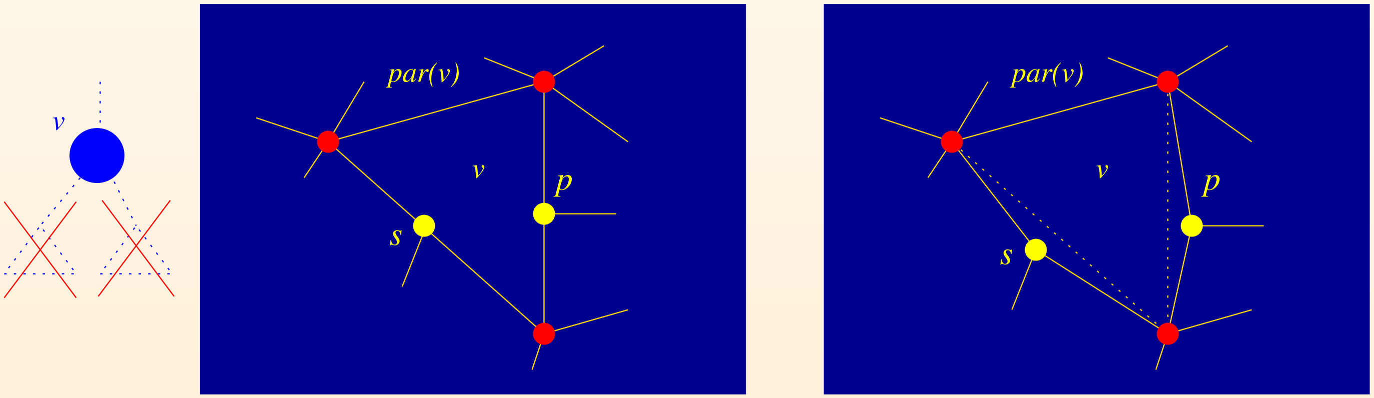
- What is a degenerate quadrilateral? How does it show up?




- Note that if $v \in T_G$ corresponds to a degenerate quadrilateral, then $v$ is either a leaf or a vertex of degree 2 of $T_G$.

# The Algorithm

- What is a degenerate pentagon? How does it show up?




- Note that if $v \in T_G$ corresponds to a degenerate pentagon, then $v$ is a leaf of $T_G$.

# The Algorithm

- How does the algorithm process the sets $V_k, V_{k-1}, \ldots, V_1, V_0$?

# The Algorithm

- How does the algorithm process the sets $V_k, V_{k-1}, \ldots, V_1, V_0$?

- Suppose the algorithm is about to process vertex set $V_i$ with $2 \leq i \leq k$. The **first step** is to eliminate ALL LEAVES of $T_G$ in $V_i \cup V_{i-1} \cup V_{i-2}$ that correspond to degenerate quadrilaterals and degenerate pentagons.

# The Algorithm

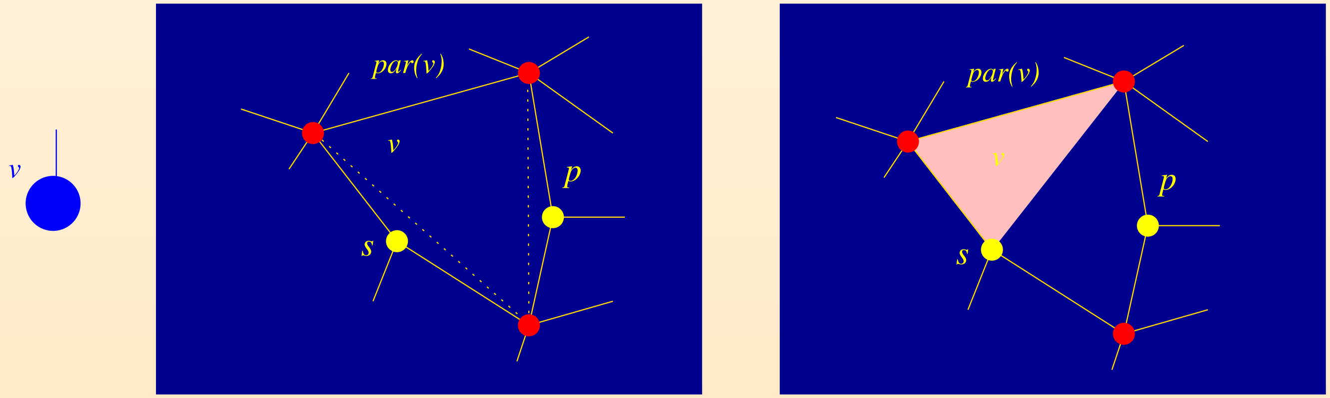
- How does the algorithm process the sets $V_k, V_{k-1}, \ldots, V_1, V_0$?

- Suppose the algorithm is about to process vertex set $V_i$ with $2 \le i \le k$. The **first step** is to eliminate ALL LEAVES of $T_G$ in $V_i \cup V_{i-1} \cup V_{i-2}$ that correspond to degenerate quadrilaterals and degenerate pentagons.

- If $v \in (V_i \cup V_{i-1} \cup V_{i-2})$ is a leaf of $T_G$ and it corresponds to a degenerate quadrilateral, then remove it from both $T_G$ and $V_i \cup V_{i-1} \cup V_{i-2}$, and output the quadrilateral corresponding to $v$.

# The Algorithm

- How does the algorithm process the sets $V_k, V_{k-1}, \ldots, V_1, V_0$?
- Suppose the algorithm is about to process vertex set $V_i$ with $2 \leq i \leq k$. The **first step** is to eliminate ALL LEAVES of $T_G$ in $V_i \cup V_{i-1} \cup V_{i-2}$ that correspond to degenerate quadrilaterals and degenerate pentagons.
- If $v \in (V_i \cup V_{i-1} \cup V_{i-2})$ is a leaf of $T_G$ and it corresponds to a degenerate quadrilateral, then remove it from both $T_G$ and $V_i \cup V_{i-1} \cup V_{i-2}$, and output the quadrilateral corresponding to $v$.
- If $v \in (V_i \cup V_{i-1} \cup V_{i-2})$ corresponds to a degenerate pentagon (and therefore it is a leaf of $T_G$), then subdivide this pentagon into a quadrilateral and a leftover triangle, $\triangle$, as shown below, output the quadrilateral, and let $v$ correspond to $\triangle$.



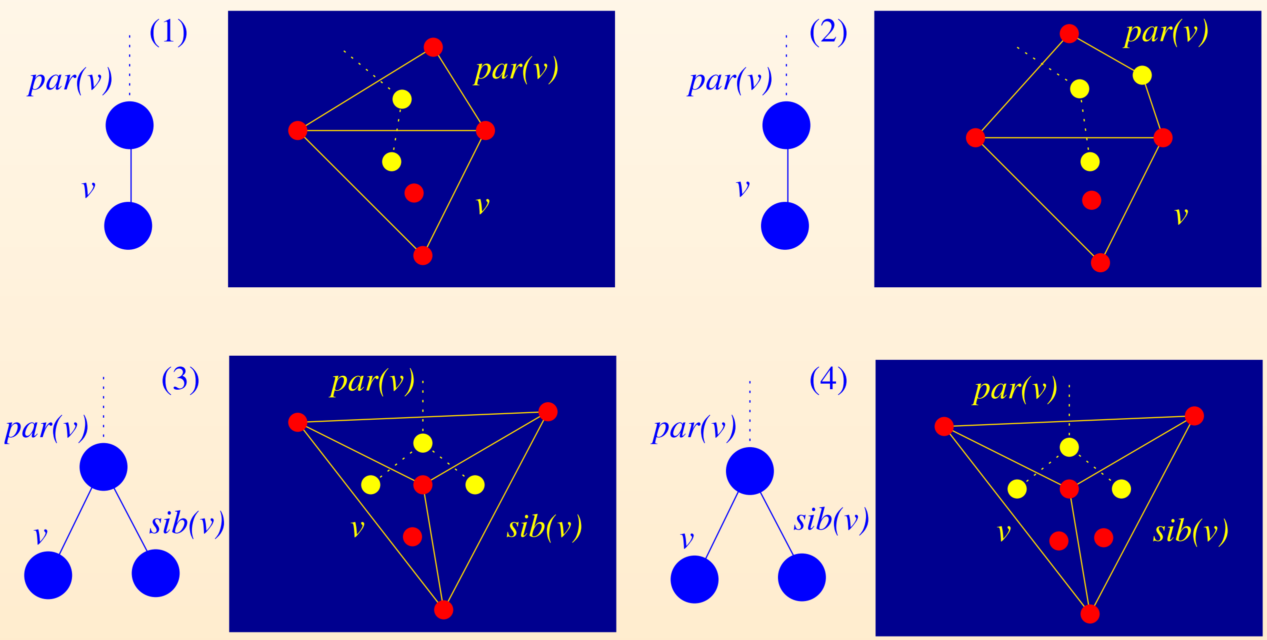
# The Algorithm

- The **second step** is to eliminate ALL LEAVES of $T_G$ in $V_i$ that correspond to non-empty triangles.

# The Algorithm

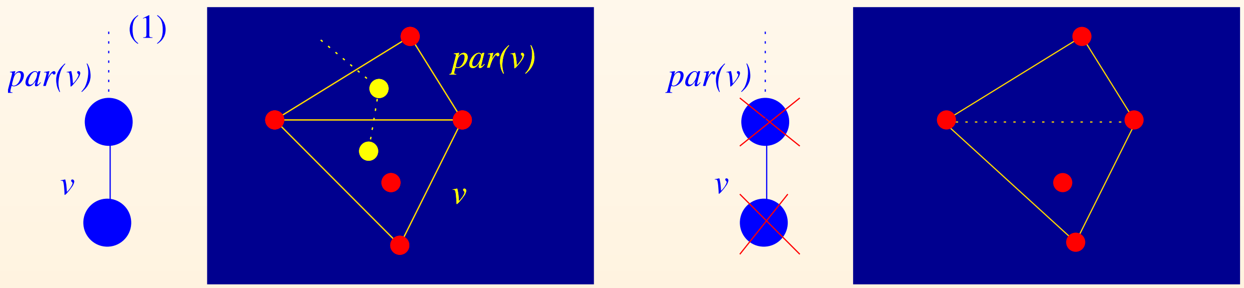
- The **second step** is to eliminate ALL LEAVES of $T_G$ in $V_i$ that correspond to non-empty triangles.

- If $v \in V_i$ corresponds to a non-empty triangle (and therefore it is a leaf of $T_G$), then we have four cases to deal with:



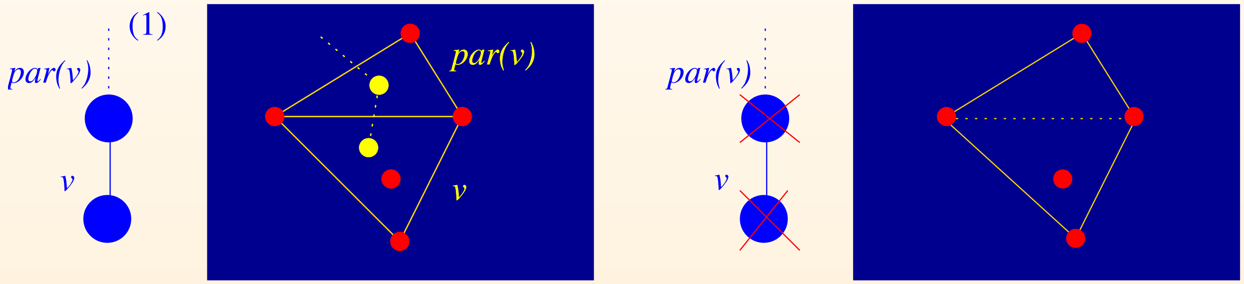
# The Algorithm

- In case (1), the algorithm combines the non-empty triangle and the triangle corresponding to $v$ and $par(v)$, respectively, in order to form a quadrilateral with an interior vertex:



# The Algorithm

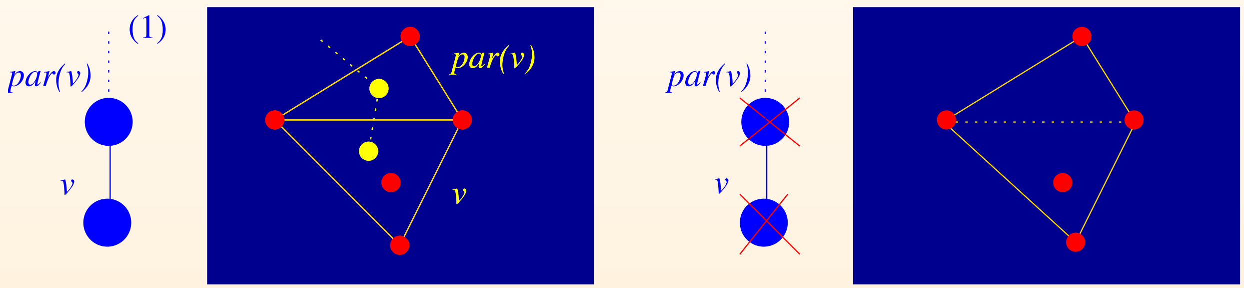
- In case (1), the algorithm combines the non-empty triangle and the triangle corresponding to $v$ and $par(v)$, respectively, in order to form a quadrilateral with an interior vertex:




- It can be shown that a quadrilateral with a vertex inside it can be decomposed into five strictly convex quadrilaterals by using three Steiner points.

# The Algorithm

- In case (1), the algorithm combines the non-empty triangle and the triangle corresponding to $v$ and $par(v)$, respectively, in order to form a quadrilateral with an interior vertex:

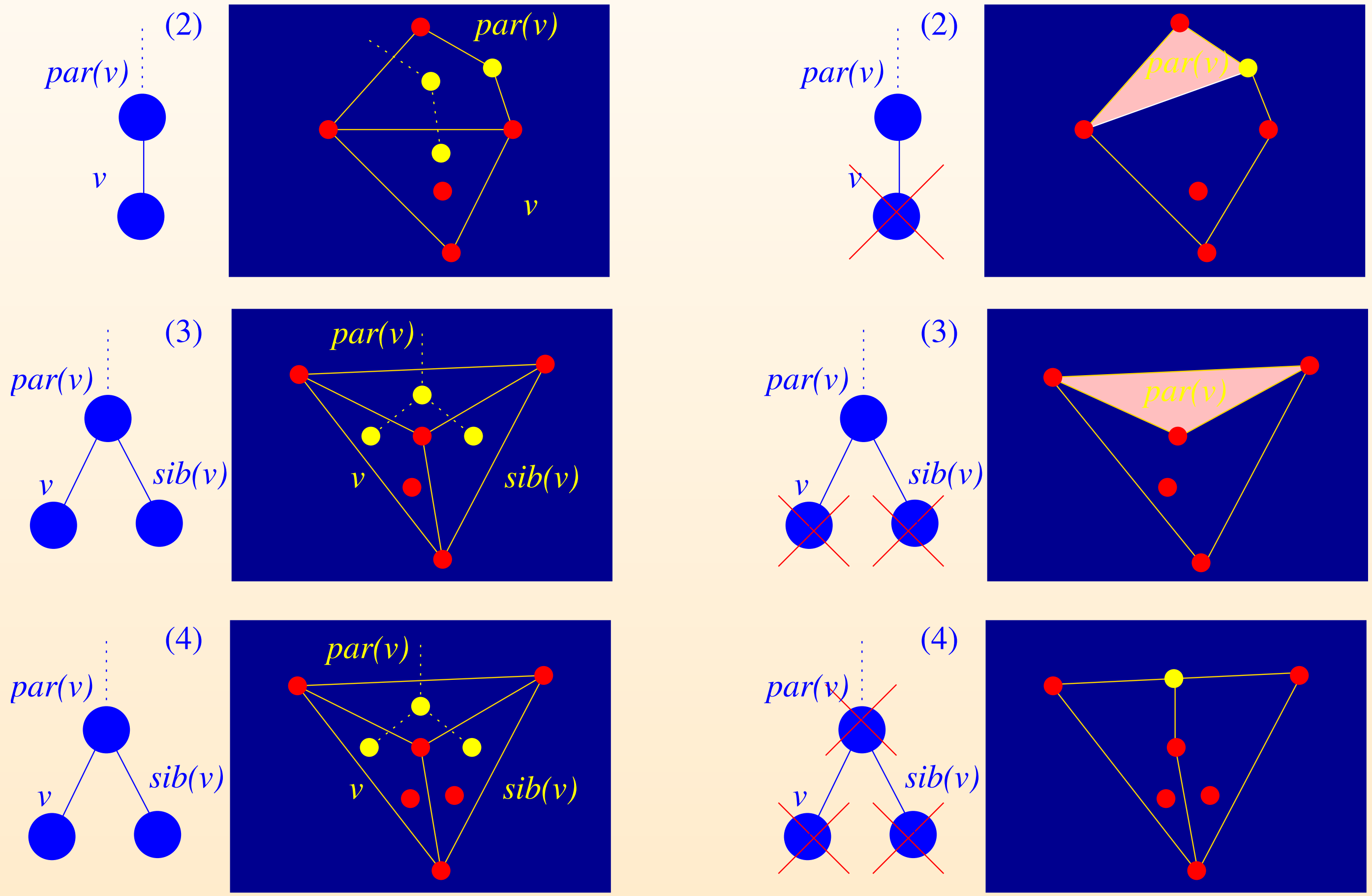



- It can be shown that a quadrilateral with a vertex inside it can be decomposed into five strictly convex quadrilaterals by using three Steiner points.

- After performing the above decomposition and outputting the resulting quadrilaterals, the algorithm removes $v$ and $par(v)$ from $T_G$ and their corresponding vertex sets.

# The Algorithm

- Cases (2), (3) and (4) can be reduced to one or two instances of case (1):



# The Algorithm

- After steps 1 and 2, all vertices in $V_i$ correspond to triangles of $\mathcal{T}$, and all leaves of $T_G$ in $V_{i-1}$ and $V_{i-2}$ correspond to either triangles of $\mathcal{T}$ or non-empty triangles.

# The Algorithm

- After steps 1 and 2, all vertices in $V_i$ correspond to triangles of $\mathcal{T}$, and all leaves of $T_G$ in $V_{i-1}$ and $V_{i-2}$ correspond to either triangles of $\mathcal{T}$ or non-empty triangles.

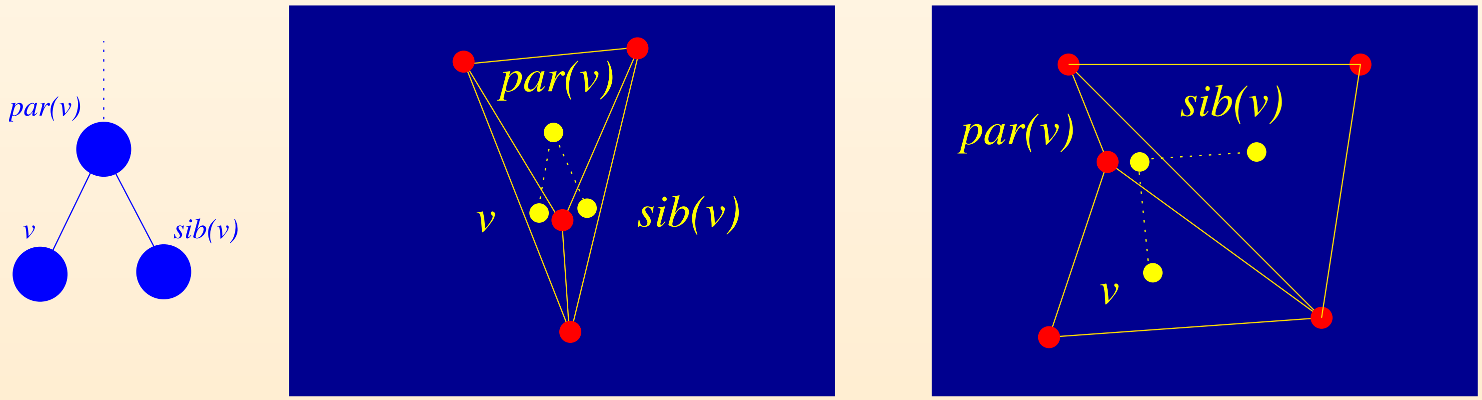
- The **third step** processes all vertices $v \in V_i$ such that $par(v)$ has two children, $v$ and its sibling, $sib(v)$.

# The Algorithm

- After steps 1 and 2, all vertices in $V_i$ correspond to triangles of $\mathcal{T}$, and all leaves of $T_G$ in $V_{i-1}$ and $V_{i-2}$ correspond to either triangles of $\mathcal{T}$ or non-empty triangles.

- The **third step** processes all vertices $v \in V_i$ such that $par(v)$ has two children, $v$ and its sibling, $sib(v)$.

- Since both $v$ and $sib(v)$ correspond to triangles of $\mathcal{T}$, the subtree rooted at $par(v)$ corresponds to either a triangle with a vertex inside it or a pentagon.



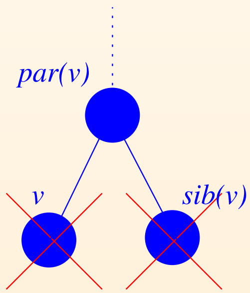

Non–empty triangle

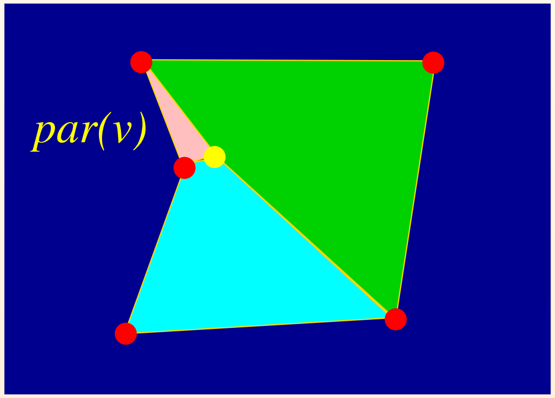
Pentagon

- We already know what to do when the subtree rooted at $par(v)$ corresponds to a triangle with a vertex inside it.

# The Algorithm

- If the triangles in $\mathcal{T}$ corresponding to $v$, $par(v)$ and $sib(v)$ form a pentagon $P$, then it can be shown that $P$ can be subdivided into two strictly convex quadrilaterals and one triangle, $\triangle$, such that $\triangle$ contains the common edge of $par(v)$ and $par(par(v))$.




- Eliminate vertices $v$ and $sib(v)$ from both $T_G$ and $V_i$, and let $par(v)$ correspond to the leftover triangle $\triangle$.

- When the vertex set $V_{i-1}$ is considered, vertex $par(v)$ is processed and the leftover triangle $\triangle$ will be combined with some other triangle(s) to form another small polygon.

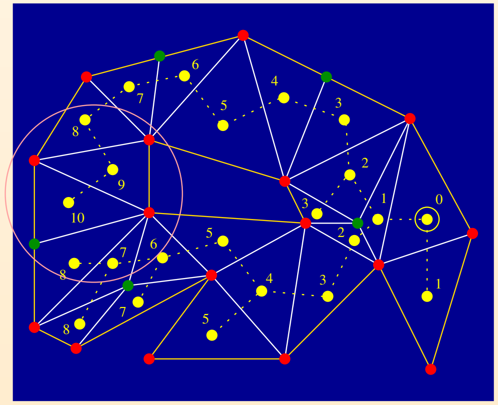
# The Algorithm

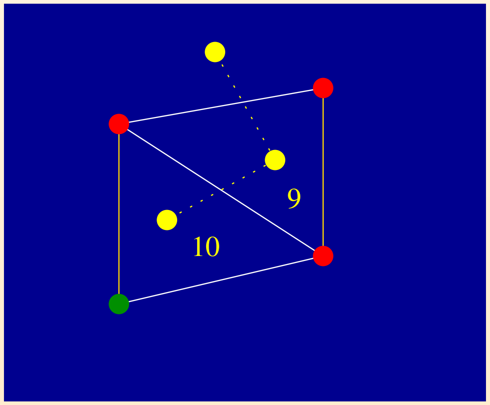
- After processing all vertices $v \in V_i$ such that $par(v)$ has two children, the algorithm starts the **fourth step** in which it processes the vertices $v \in V_i$ such that $v$ is the only child of $par(v)$ with $2 \leq i \leq k$.

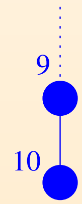
# The Algorithm

- After processing all vertices $v \in V_i$ such that $par(v)$ has two children, the algorithm starts the **fourth step** in which it processes the vertices $v \in V_i$ such that $v$ is the only child of $par(v)$ with $2 \leq i \leq k$.

- If $par(v)$ corresponds to a triangle and this triangle forms a strictly convex quadrilateral with the triangle corresponding to $v$, then the algorithm outputs the quadrilateral and removes both $v$ and $par(v)$ from $T_G$ and from their corresponding vertex sets.





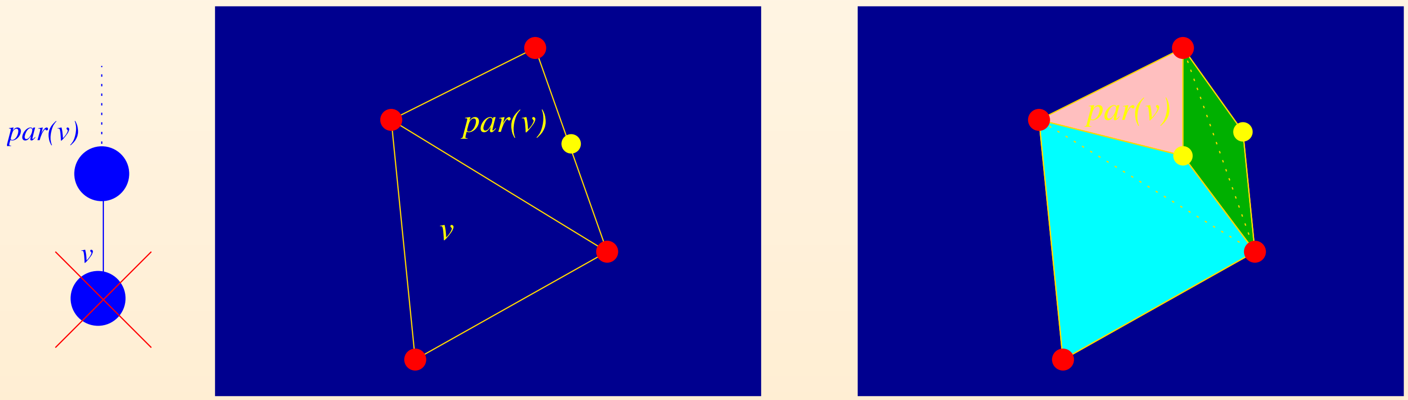
# The Algorithm

- If $par(v)$ corresponds to a degenerate quadrilateral, then this quadrilateral can be combined with the triangle corresponding to $v$ to form a pentagon $P$.

# The Algorithm

- If $par(v)$ corresponds to a degenerate quadrilateral, then this quadrilateral can be combined with the triangle corresponding to $v$ to form a pentagon $P$.

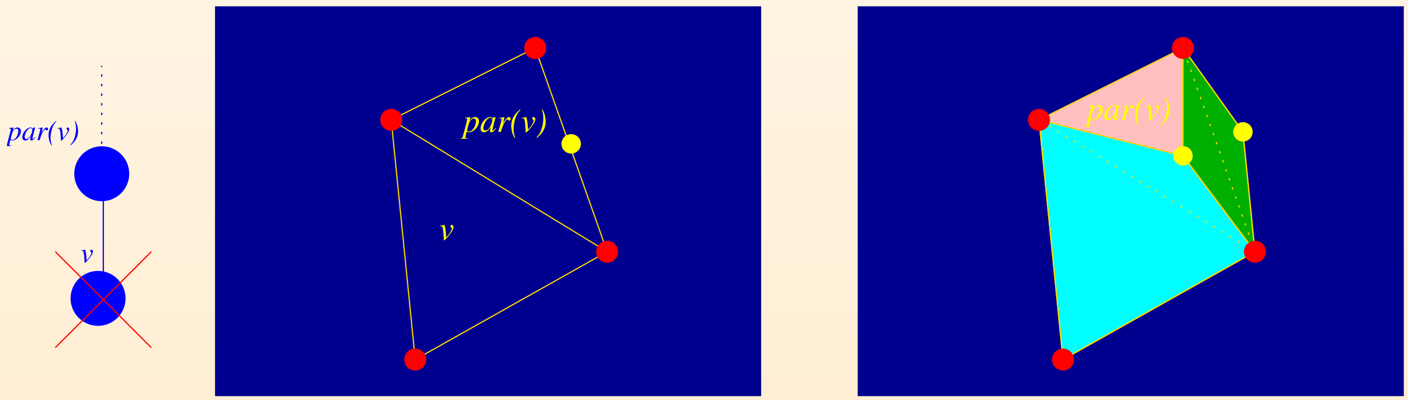
- It can be shown that the pentagon $P$ can be decomposed into two strictly convex quadrilaterals and one leftover triangle, $\triangle$, by adding one Steiner point inside the degenerate quadrilateral. Furthermore, the triangle $\triangle$ contains the common edge of $par(v)$ and $par(par(v))$.



# The Algorithm

- If $par(v)$ corresponds to a degenerate quadrilateral, then this quadrilateral can be combined with the triangle corresponding to $v$ to form a pentagon $P$.

- It can be shown that the pentagon $P$ can be decomposed into two strictly convex quadrilaterals and one leftover triangle, $\triangle$, by adding one Steiner point inside the degenerate quadrilateral. Furthermore, the triangle $\triangle$ contains the common edge of $par(v)$ and $par(par(v))$.




- After performing the above decomposition, the algorithm makes $par(v)$ correspond to the leftover triangle $\triangle$, removes $v$ from both $T_G$ and $V_i$, and outputs the resulting quadrilaterals.

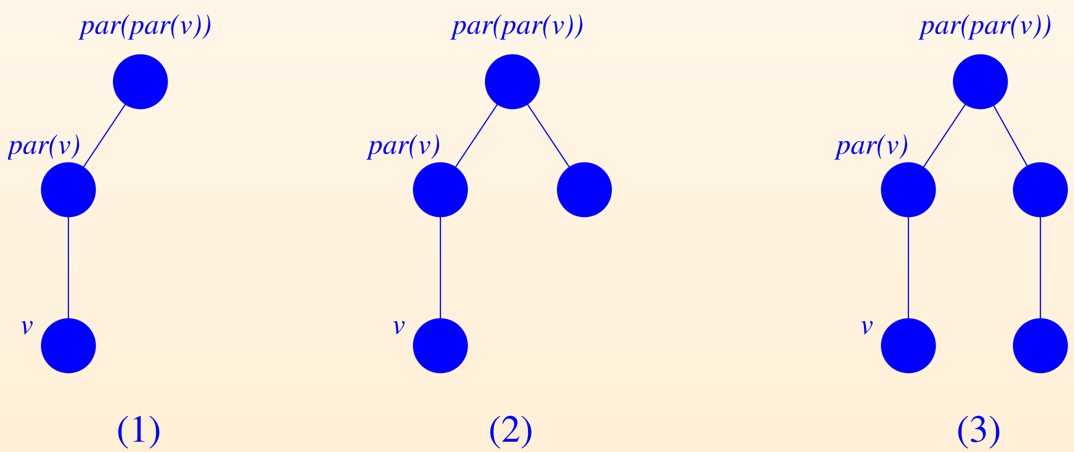
# The Algorithm

- If both $v$ and $par(v)$ correspond to triangles of $\mathcal{T}$, but these triangles DO NOT form a strictly convex quadrilateral, we consider the subtree of $T_G$ rooted at $par(par(v))$.

# The Algorithm

- If both $v$ and $par(v)$ correspond to triangles of $\mathcal{T}$, but these triangles DO NOT form a strictly convex quadrilateral, we consider the subtree of $T_G$ rooted at $par(par(v))$.

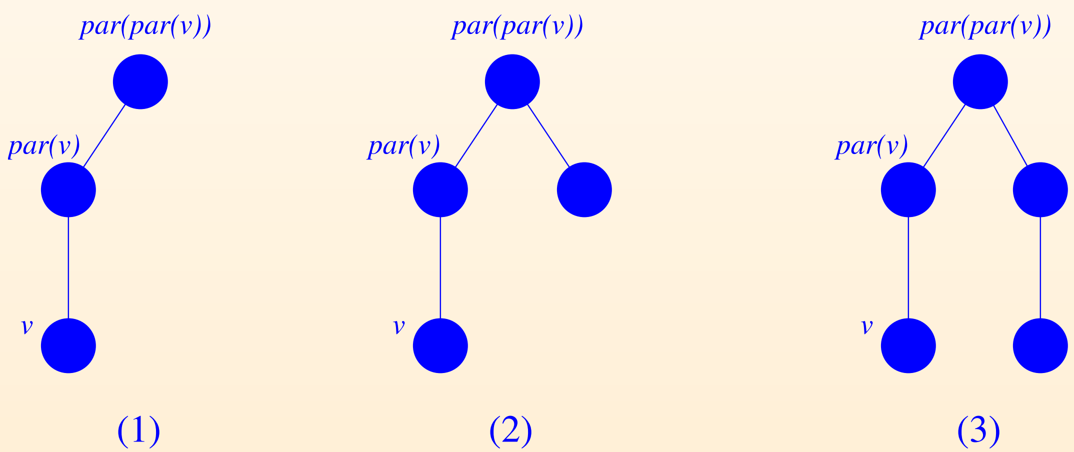
- Up to isomorphism, we only have the following three possible subtrees rooted at $par(par(v))$:



# The Algorithm

- If both $v$ and $par(v)$ correspond to triangles of $\mathcal{T}$, but these triangles DO NOT form a strictly convex quadrilateral, we consider the subtree of $T_G$ rooted at $par(par(v))$.

- Up to isomorphism, we only have the following three possible subtrees rooted at $par(par(v))$:



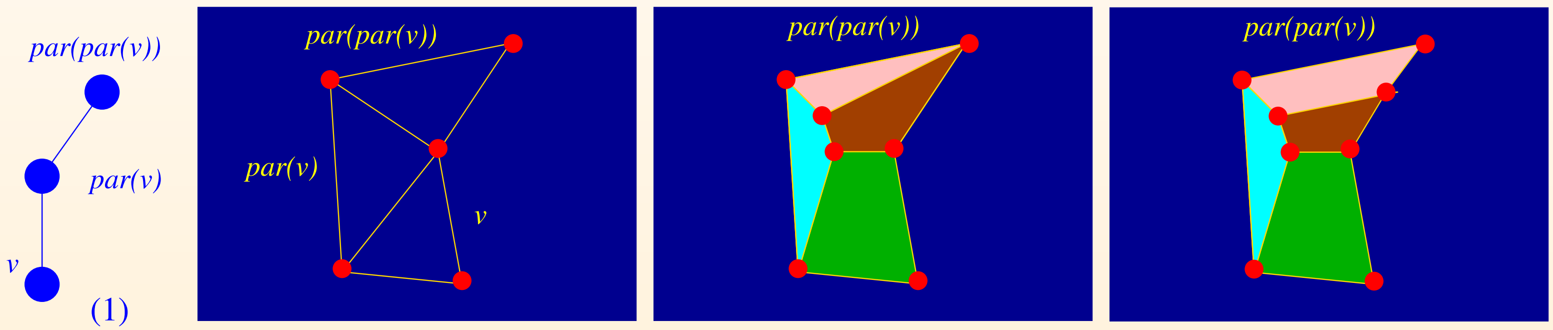

- The trees above correspond to (1) a pentagon or hexagon, (2) a quadrilateral with one or two vertices inside it, a hexagon, or a hexagon with a vertex inside it, and (3) a triangle with two vertices inside it, a pentagon with a vertex inside it, or a septagon.

# The Algorithm

- In case (1), if $par(par(v))$ corresponds to a triangle, then the union of the triangles corresponding to $v$, $par(v)$ and $par(par(v))$ forms a pentagon $P$.

# The Algorithm

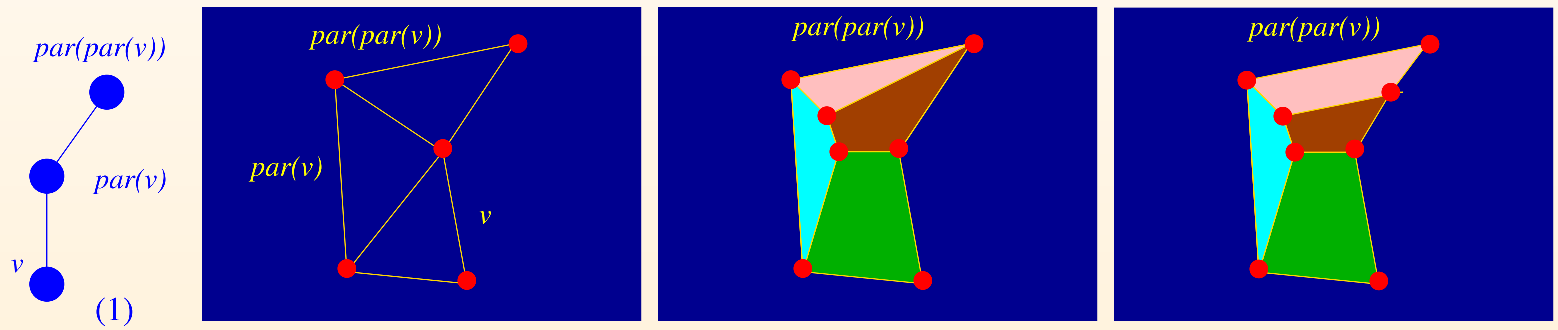
- In case (1), if $par(par(v))$ corresponds to a triangle, then the union of the triangles corresponding to $v$, $par(v)$ and $par(par(v))$ forms a pentagon $P$.
- It can be shown that the pentagon $P$ can be decomposed into either two strictly convex quadrilaterals and a leftover triangle or three quadrilaterals.



# The Algorithm

- In case (1), if $par(par(v))$ corresponds to a triangle, then the union of the triangles corresponding to $v$, $par(v)$ and $par(par(v))$ forms a pentagon $P$.

- It can be shown that the pentagon $P$ can be decomposed into either two strictly convex quadrilaterals and a leftover triangle or three quadrilaterals.

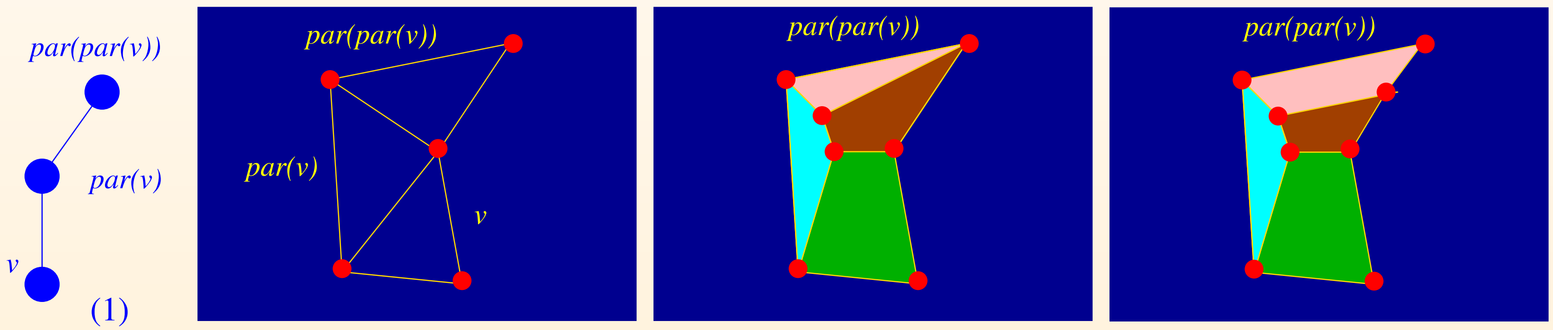



- In the former case, two Steiner points are added, and $v$ and $par(v)$ get removed from $T_G$ and their corresponding vertex sets. In the latter case, three Steiner points are added and $v$, $par(v)$ and $par(par(v))$ get removed from $T_G$ and their corresponding vertex sets.

# The Algorithm

- In case (1), if $par(par(v))$ corresponds to a triangle, then the union of the triangles corresponding to $v$, $par(v)$ and $par(par(v))$ forms a pentagon $P$.

- It can be shown that the pentagon $P$ can be decomposed into either two strictly convex quadrilaterals and a leftover triangle or three quadrilaterals.

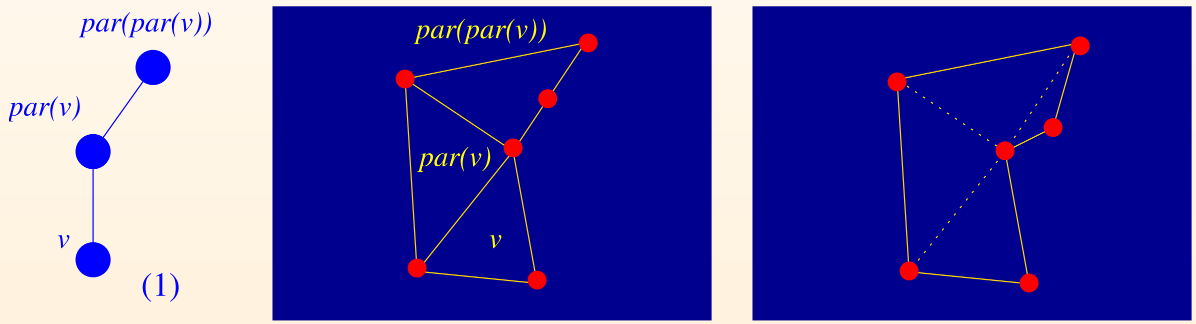



- In the former case, two Steiner points are added, and $v$ and $par(v)$ get removed from $T_G$ and their corresponding vertex sets. In the latter case, three Steiner points are added and $v$, $par(v)$ and $par(par(v))$ get removed from $T_G$ and their corresponding vertex sets.

- The algorithm chooses either decomposition based on the position of the edge shared by the triangles corresponding to $par(par(v))$ and its parent, if any. If there is no $par(par(par(v)))$ the latter decomposition is chosen.

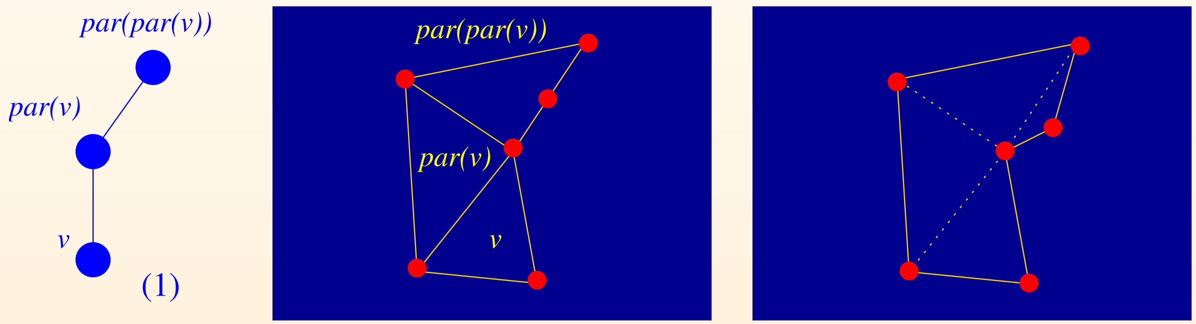
# The Algorithm

- In case (1), if $par(par(v))$ corresponds to a degenerate quadrilateral, then the union of the triangles corresponding to $v$, $par(v)$ and $par(par(v))$ forms a hexagon $H$.



# The Algorithm

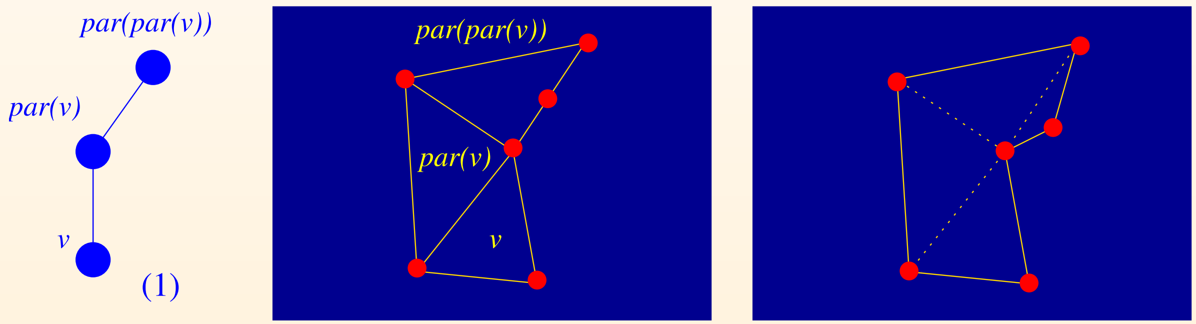
- In case (1), if $par(par(v))$ corresponds to a degenerate quadrilateral, then the union of the triangles corresponding to $v$, $par(v)$ and $par(par(v))$ forms a hexagon $H$.




- It can be shown that the hexagon $H$ can be decomposed into at most four strictly convex quadrilaterals by adding at most three Steiner points.

# The Algorithm

- In case (1), if $par(par(v))$ corresponds to a degenerate quadrilateral, then the union of the triangles corresponding to $v$, $par(v)$ and $par(par(v))$ forms a hexagon $H$.

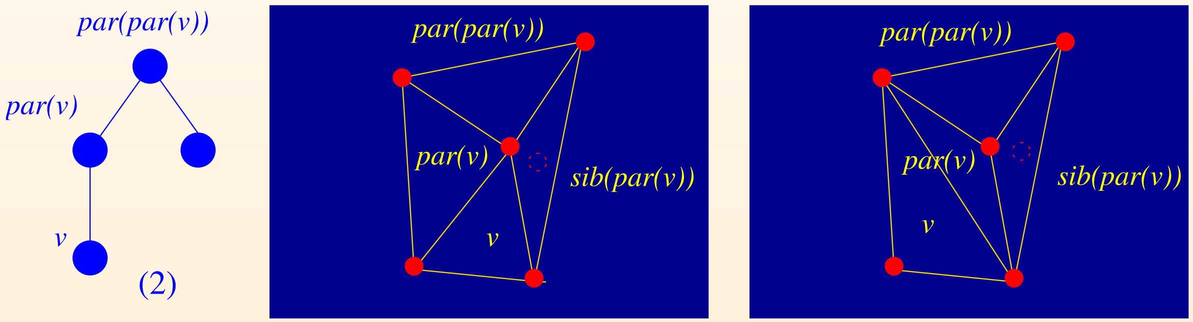



- It can be shown that the hexagon $H$ can be decomposed into at most four strictly convex quadrilaterals by adding at most three Steiner points.

- After performing the above decomposition, the algorithm removes $v$, $par(v)$, and $par(par(v))$ from $T_G$ and from their corresponding vertex sets.

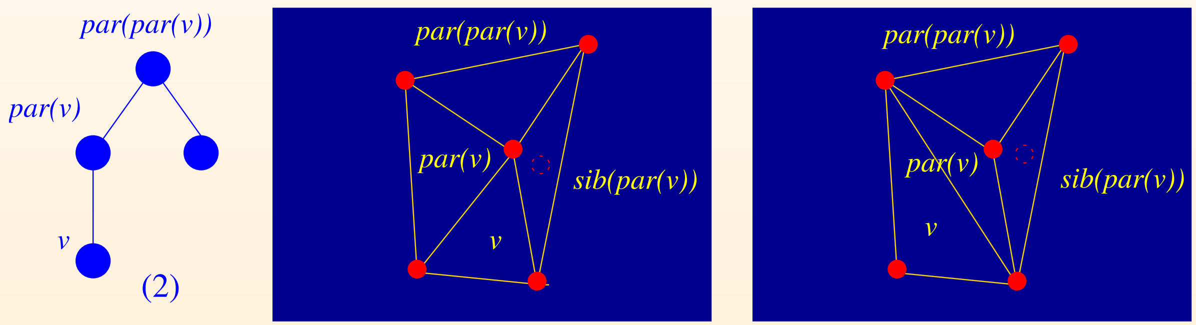
# The Algorithm

- In case (2), if either the triangle corresponding to $v$ or the triangle corresponding to $par(v)$ shares an edge with the triangle corresponding to $sib(par(v))$, the subtree rooted at $par(par(v))$ corresponds to a quadrilateral $Q$ with one or two vertices inside it.



# The Algorithm

- In case (2), if either the triangle corresponding to $v$ or the triangle corresponding to $par(v)$ shares an edge with the triangle corresponding to $sib(par(v))$, the subtree rooted at $par(par(v))$ corresponds to a quadrilateral $Q$ with one or two vertices inside it.




- The number of vertices inside $Q$ is determined by the entity corresponding to $sib(par(v))$: it is one vertex if $sib(par(v))$ corresponds to a triangle and two vertices if it corresponds to a non-empty triangle.

# The Algorithm

- If there is only one vertex inside $Q$, it can be shown that $Q$ can be decomposed into at most five strictly convex quadrilaterals by adding at most three Steiner points. Otherwise, $Q$ can be decomposed into at most nine strictly convex quadrilaterals by adding at most six Steiner points.

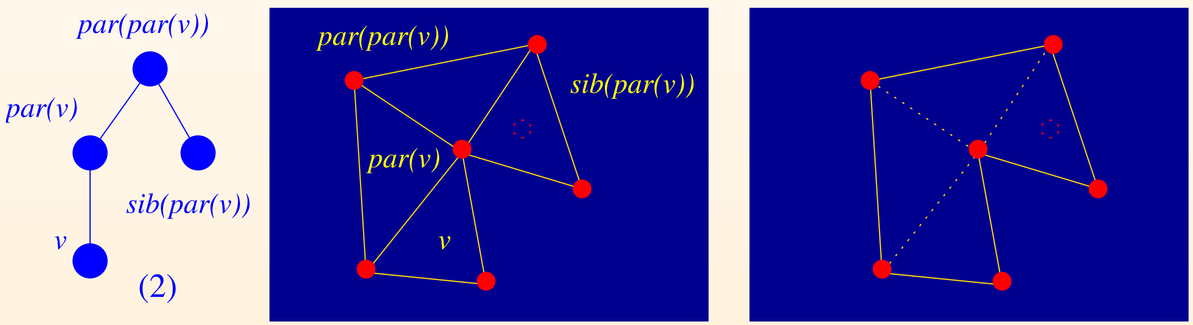
# The Algorithm

- If there is only one vertex inside $Q$, it can be shown that $Q$ can be decomposed into at most five strictly convex quadrilaterals by adding at most three Steiner points. Otherwise, $Q$ can be decomposed into at most nine strictly convex quadrilaterals by adding at most six Steiner points.

- In either case, the vertices $v$, $par(v)$, $sib(par(v))$, and $par(par(v))$ get removed from $T_G$ and their corresponding vertex sets.

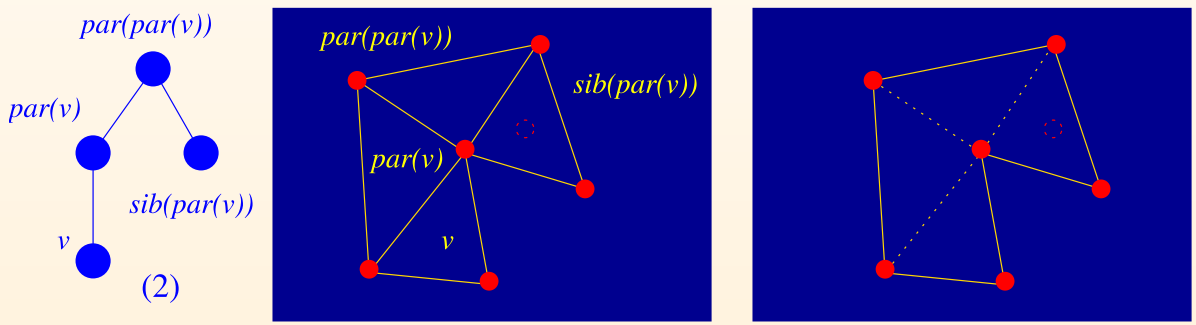
# The Algorithm

- In case (2), if neither the triangle corresponding to $v$ nor the triangle corresponding to $par(v)$ shares an edge with the triangle corresponding to $sib(par(v))$, the subtree rooted at $par(par(v))$ corresponds to either a hexagon or a hexagon with a vertex inside it.



# The Algorithm

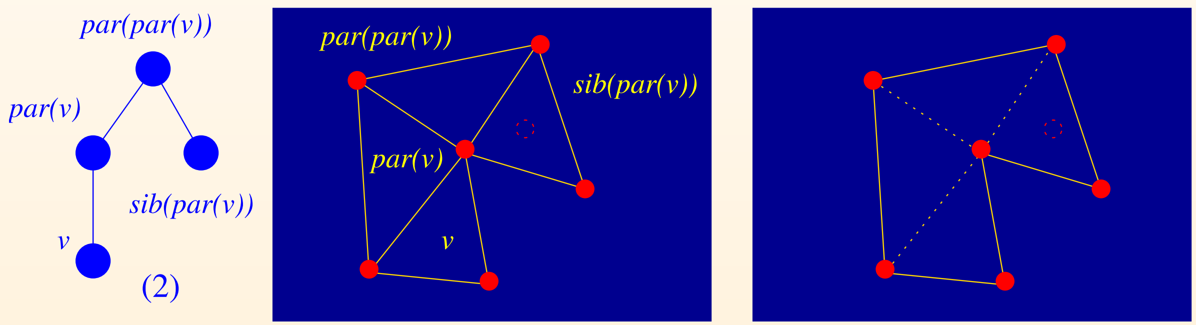
- In case (2), if neither the triangle corresponding to $v$ nor the triangle corresponding to $par(v)$ shares an edge with the triangle corresponding to $sib(par(v))$, the subtree rooted at $par(par(v))$ corresponds to either a hexagon or a hexagon with a vertex inside it.




- The former case can be reduced to case (1) (with $par(par(v))$ being a degenerate quadrilateral), and the latter case is analogous to the case in which the subtree rooted at $par(par(v))$ corresponds to a quadrilateral with two vertices inside it.

# The Algorithm

- In case (2), if neither the triangle corresponding to $v$ nor the triangle corresponding to $par(v)$ shares an edge with the triangle corresponding to $sib(par(v))$, the subtree rooted at $par(par(v))$ corresponds to either a hexagon or a hexagon with a vertex inside it.

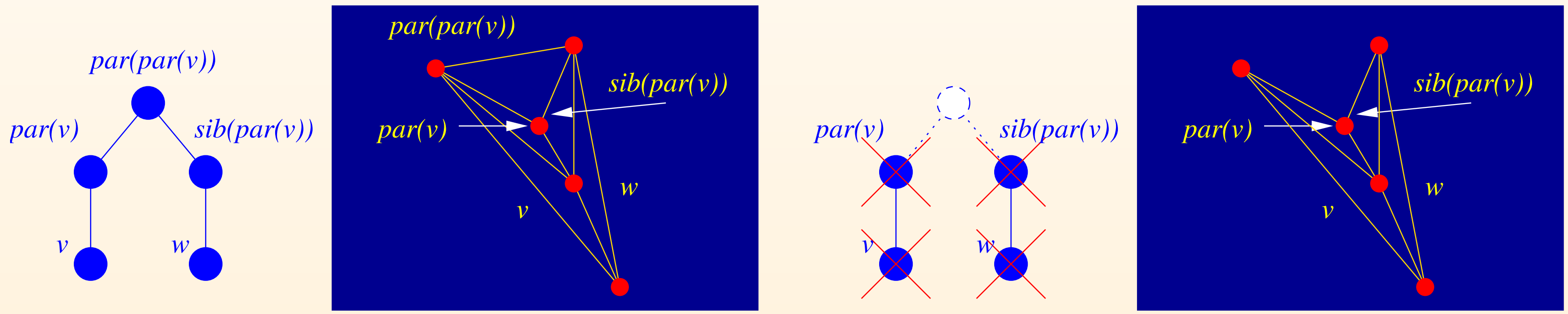



- The former case can be reduced to case (1) (with $par(par(v))$ being a degenerate quadrilateral), and the latter case is analogous to the case in which the subtree rooted at $par(par(v))$ corresponds to a quadrilateral with two vertices inside it.

- In either case, the $v$, $par(v)$, $sib(par(v))$, and $par(par(v))$ get removed from $T_G$ and their corresponding vertex sets.

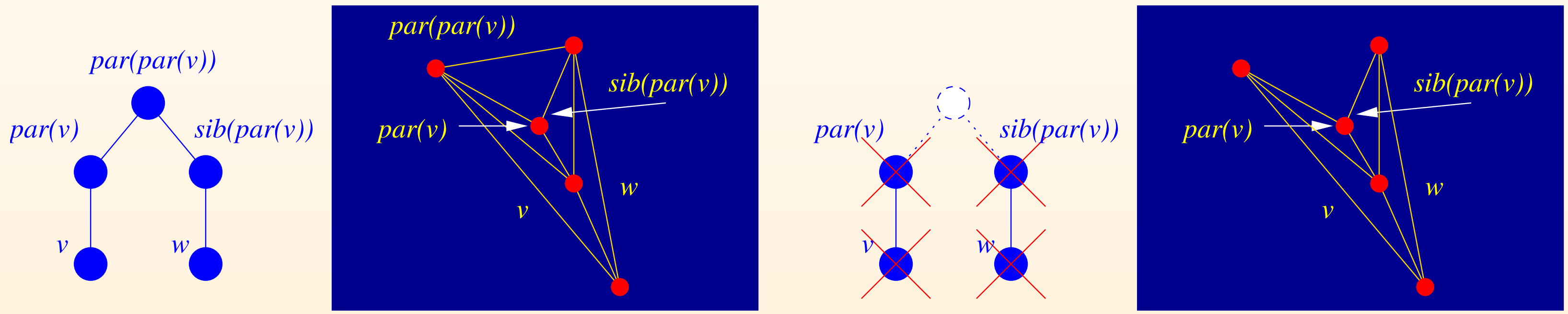
# The Algorithm

- In case (3), if the triangles corresponding to $v$ and $par(v)$, or only one of them, are adjacent to both the triangles corresponding to $sib(par(v))$ and its child, then the subtree rooted at $par(par(v))$ corresponds to a triangle with two vertices inside it.



# The Algorithm

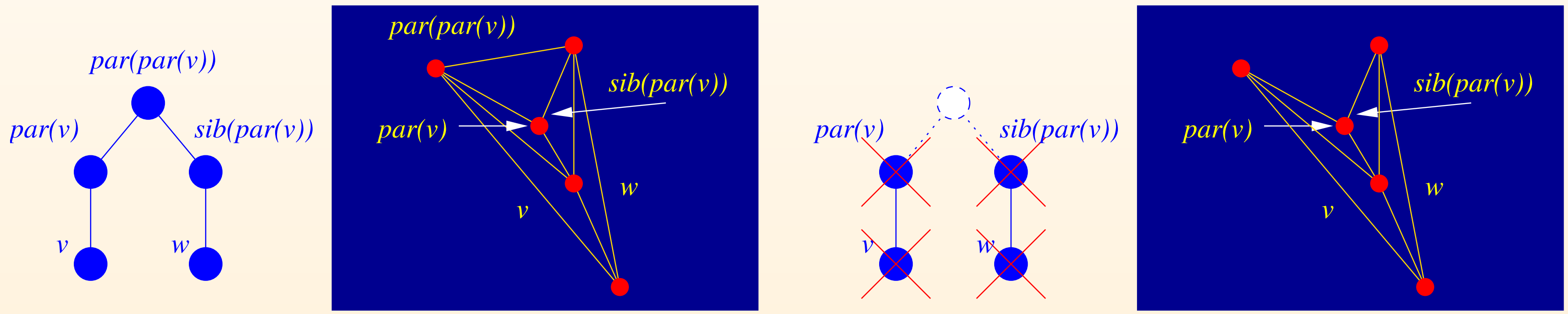
- In case (3), if the triangles corresponding to $v$ and $par(v)$, or only one of them, are adjacent to both the triangles corresponding to $sib(par(v))$ and its child, then the subtree rooted at $par(par(v))$ corresponds to a triangle with two vertices inside it.




- As shown in the above figure, we can deal with this case by considering the quadrilateral with one vertex inside it, which corresponds to the vertices $v$, $par(v)$, $sib(par(v))$ and the child of $sib(par(v))$.

# The Algorithm

- In case (3), if the triangles corresponding to $v$ and $par(v)$, or only one of them, are adjacent to both the triangles corresponding to $sib(par(v))$ and its child, then the subtree rooted at $par(par(v))$ corresponds to a triangle with two vertices inside it.

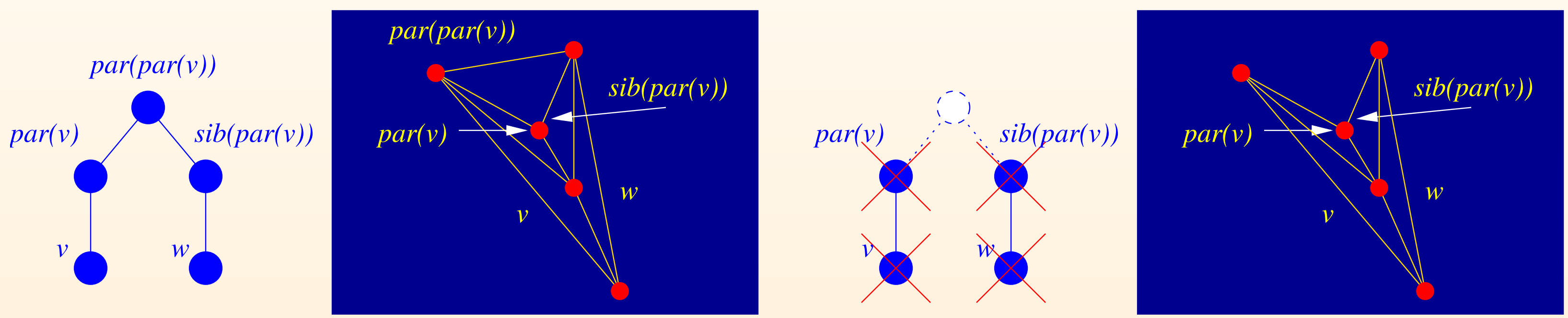



- As shown in the above figure, we can deal with this case by considering the quadrilateral with one vertex inside it, which corresponds to the vertices $v$, $par(v)$, $sib(par(v))$ and the child of $sib(par(v))$.

- We already know how to deal with a quadrilateral with a vertex inside it.

# The Algorithm

- In case (3), if the triangles corresponding to $v$ and $par(v)$, or only one of them, are adjacent to both the triangles corresponding to $sib(par(v))$ and its child, then the subtree rooted at $par(par(v))$ corresponds to a triangle with two vertices inside it.



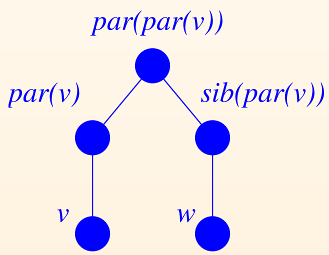

- As shown in the above figure, we can deal with this case by considering the quadrilateral with one vertex inside it, which corresponds to the vertices $v$, $par(v)$, $sib(par(v))$ and the child of $sib(par(v))$.

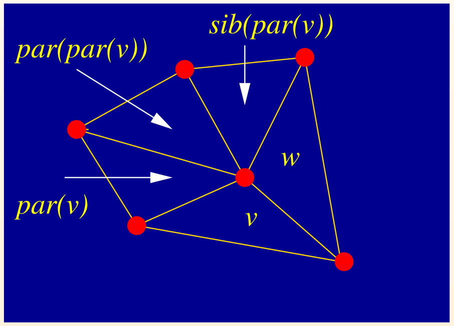
- We already know how to deal with a quadrilateral with a vertex inside it.

- The vertices $v$, $par(v)$, $sib(par(v))$ and the child of $sib(par(v))$ get removed from $T_G$ and from their corresponding vertex sets.

# The Algorithm

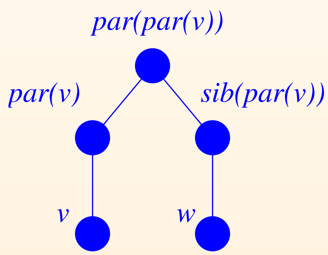
- In case (3), if the triangle corresponding to $v$ (resp. $par(v)$) has only one edge in common with either $sib(par(v))$ or its child, and the triangle corresponding to $par(v)$ (resp. $v$) shares no edge with $sib(par(v))$ nor its child, then the subtree rooted at $par(par(v))$ corresponds to a pentagon $P$ with one vertex inside it.

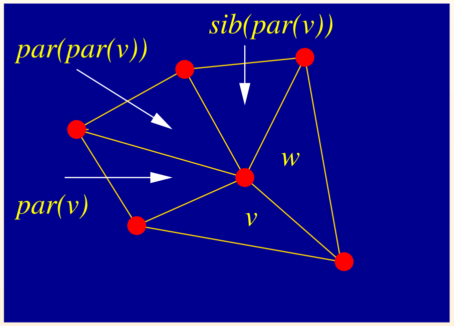


# The Algorithm

- In case (3), if the triangle corresponding to $v$ (resp. $par(v)$) has only one edge in common with either $sib(par(v))$ or its child, and the triangle corresponding to $par(v)$ (resp. $v$) shares no edge with $sib(par(v))$ nor its child, then the subtree rooted at $par(par(v))$ corresponds to a pentagon $P$ with one vertex inside it.

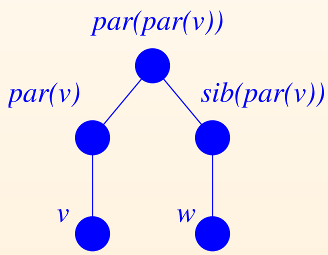


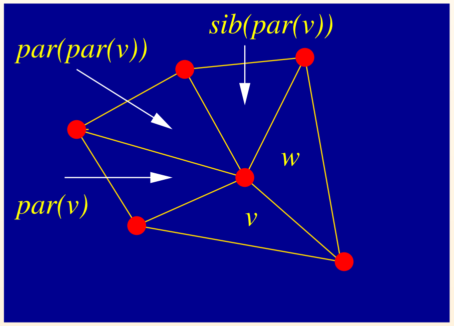




- It can be shown that the pentagon $P$ can be decomposed into at most four strictly convex quadrilaterals and a leftover triangle, $\triangle$, by adding at most three Steiner points inside $P$. The leftover triangle $\triangle$ contains the edge of $par(par(v))$ that is not adjacent to $par(v)$ nor to $sib(par(v))$.

# The Algorithm

- In case (3), if the triangle corresponding to $v$ (resp. $par(v)$) has only one edge in common with either $sib(par(v))$ or its child, and the triangle corresponding to $par(v)$ (resp. $v$) shares no edge with $sib(par(v))$ nor its child, then the subtree rooted at $par(par(v))$ corresponds to a pentagon $P$ with one vertex inside it.



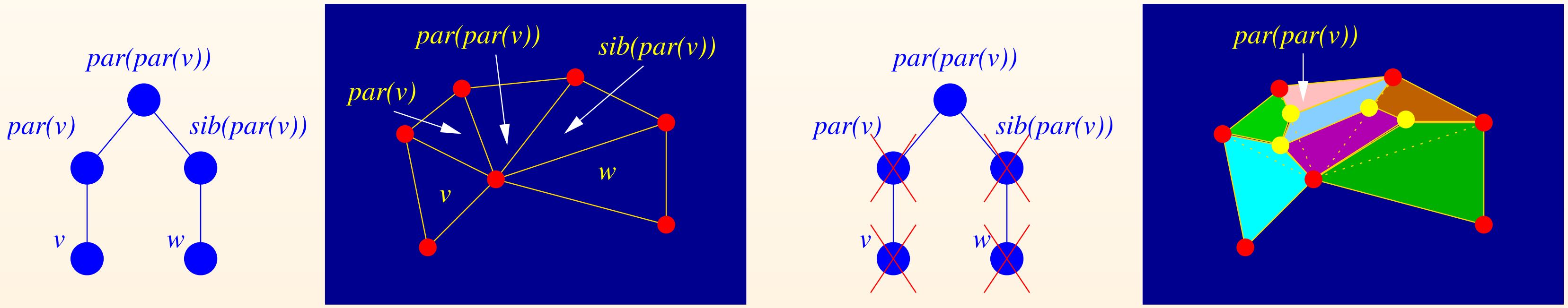




- It can be shown that the pentagon $P$ can be decomposed into at most four strictly convex quadrilaterals and a leftover triangle, $\triangle$, by adding at most three Steiner points inside $P$. The leftover triangle $\triangle$ contains the edge of $par(par(v))$ that is not adjacent to $par(v)$ nor to $sib(par(v))$.

- The algorithm removes $v$, $par(v)$, $sib(par(v))$ and the child of $sib(par(v))$ from $T_G$ and from their corresponding vertex sets, and makes $par(par(v))$ correspond to $\triangle$.

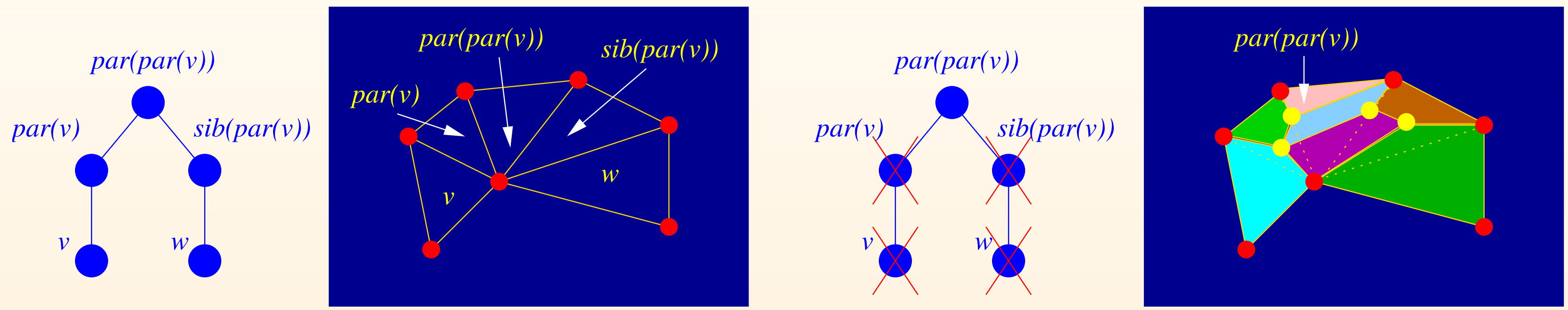
# The Algorithm

- In case (3), if neither the triangle corresponding to $v$ nor the one corresponding to $par(v)$ shares an edge with neither $sib(par(v))$ nor its child, then the subtree rooted at $par(par(v))$ corresponds to a septagon $S$.



# The Algorithm

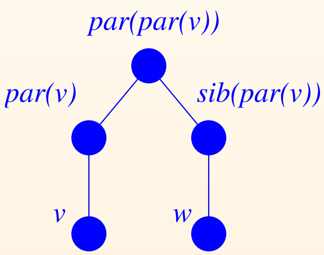
- In case (3), if neither the triangle corresponding to $v$ nor the one corresponding to $par(v)$ shares an edge with neither $sib(par(v))$ nor its child, then the subtree rooted at $par(par(v))$ corresponds to a septagon $S$.

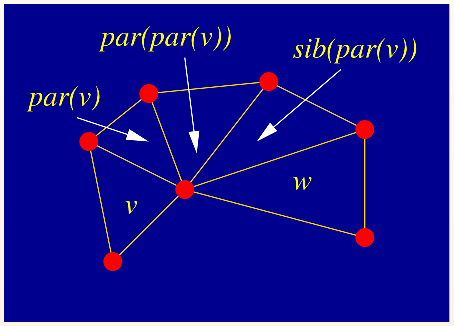


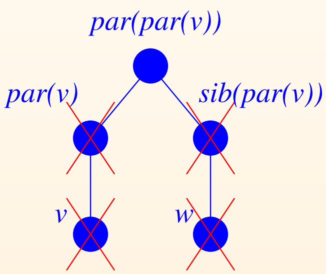

- It can be shown that the septagon $S$ can be decomposed into at most six strictly convex quadrilaterals and a leftover triangle, $\triangle$, by adding at most four Steiner points inside $S$.

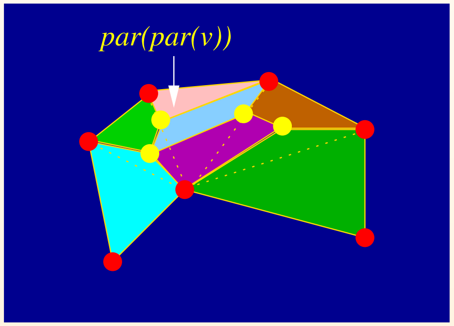
# The Algorithm

- In case (3), if neither the triangle corresponding to $v$ nor the one corresponding to $par(v)$ shares an edge with neither $sib(par(v))$ nor its child, then the subtree rooted at $par(par(v))$ corresponds to a septagon $S$.







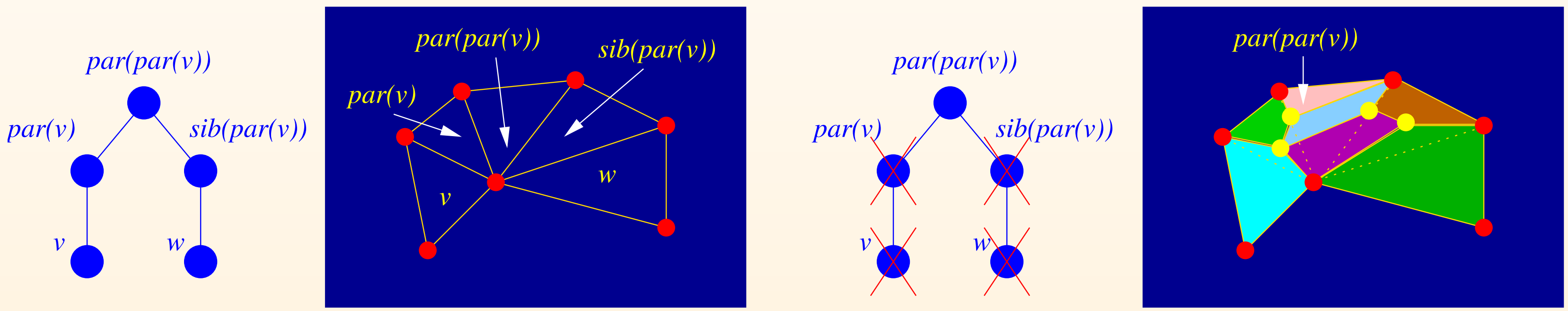



- It can be shown that the septagon $S$ can be decomposed into at most six strictly convex quadrilaterals and a leftover triangle, $\triangle$, by adding at most four Steiner points inside $S$.

- The leftover triangle $\triangle$ contains the edge of $par(par(v))$ that is not adjacent to $par(v)$ nor to $sib(par(v))$.

# The Algorithm

- In case (3), if neither the triangle corresponding to $v$ nor the one corresponding to $par(v)$ shares an edge with neither $sib(par(v))$ nor its child, then the subtree rooted at $par(par(v))$ corresponds to a septagon $S$.



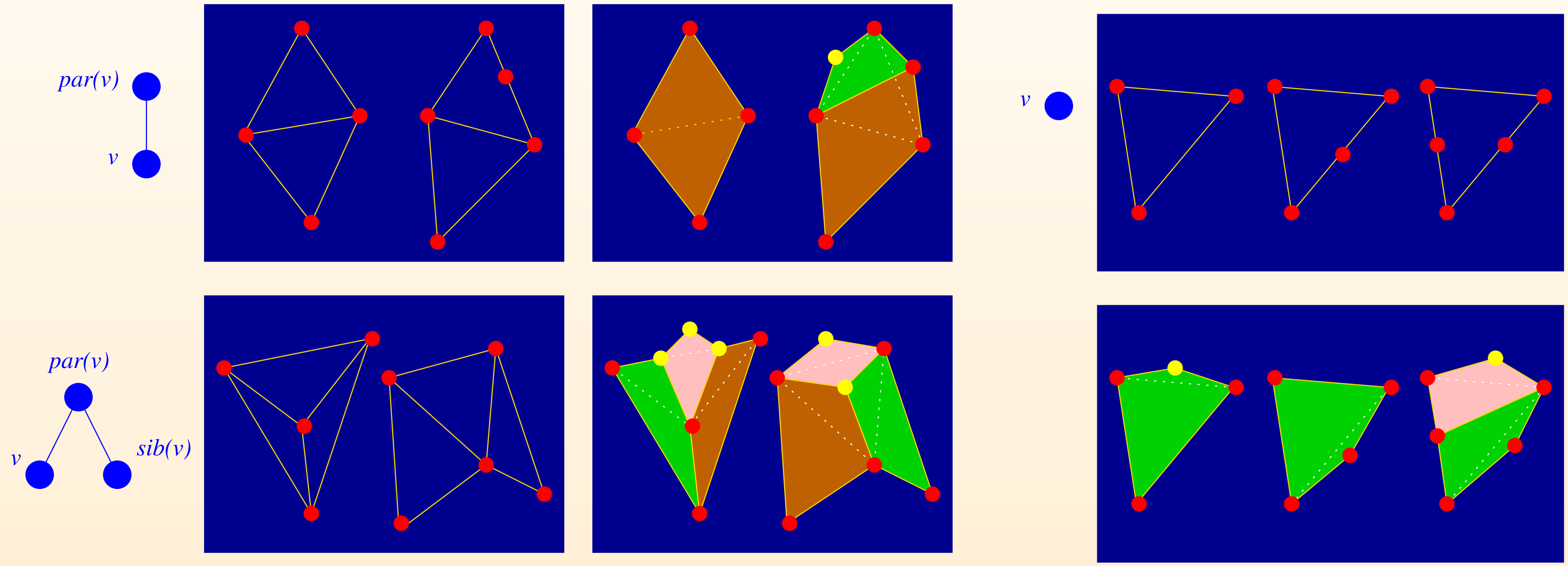

- It can be shown that the septagon $S$ can be decomposed into at most six strictly convex quadrilaterals and a leftover triangle, $\triangle$, by adding at most four Steiner points inside $S$.

- The leftover triangle $\triangle$ contains the edge of $par(par(v))$ that is not adjacent to $par(v)$ nor to $sib(par(v))$.

- The algorithm removes $v$, $par(v)$, $sib(par(v))$ and the child of $sib(par(v))$ from $T_G$ and from their corresponding vertex sets, and makes $par(par(v))$ correspond to $\triangle$.

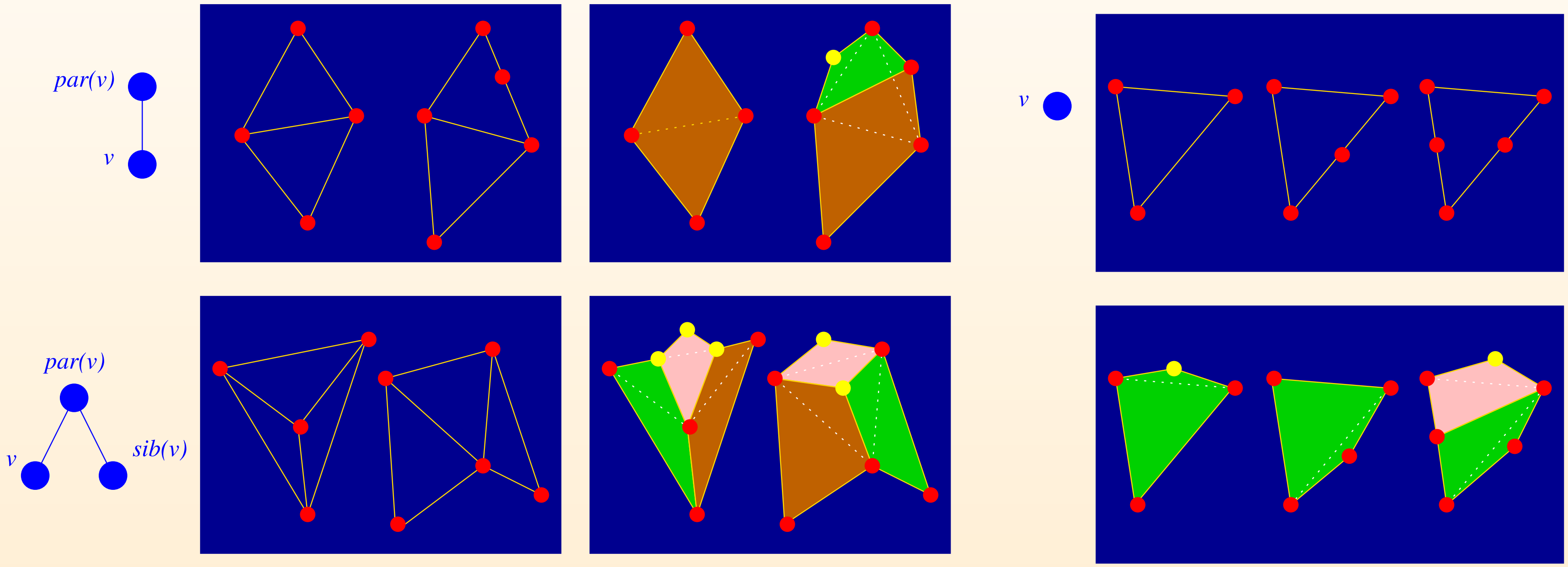
# The Algorithm

- After processing all vertices $v \in V_i$ such that $par(v)$ has only one child, the algorithm starts the **fifth and last step** in which it processes the vertex sets $V_1$ and $V_0$.



# The Algorithm

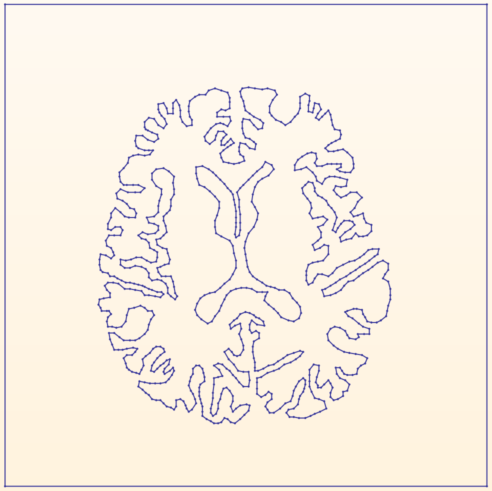
- After processing all vertices $v \in V_i$ such that $par(v)$ has only one child, the algorithm starts the **fifth and last step** in which it processes the vertex sets $V_1$ and $V_0$.




- By carefully analyzing all previous cases, it can be shown that the algorithm generates at most $\lfloor \frac{3}{2}t \rfloor$ strictly convex quadrilaterals and uses at most $t + 2$ Steiner points, where $t$ is the number of triangles in $\mathcal{T}$. Furthermore, its time and space complexity are $\mathcal{O}(t)$.

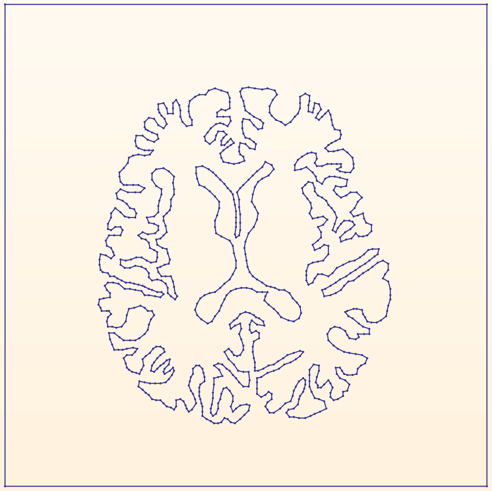
# Constrained Quadrangulations

- Our algorithm can be extended in a straightforward manner to deal with arbitrary constrained triangulations.



# Constrained Quadrangulations

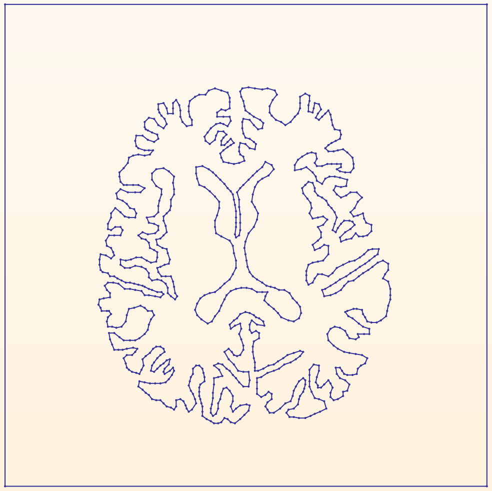
- Our algorithm can be extended in a straightforward manner to deal with arbitrary constrained triangulations.



- Given a constrained triangulation $\mathcal{T}$, build the dual graph $G$ of $\mathcal{T}$ respecting the constrained edges of $\mathcal{T}$. That is, the graph $G$ does not contain the dual edges of constrained edges of $\mathcal{T}$.

# Constrained Quadrangulations

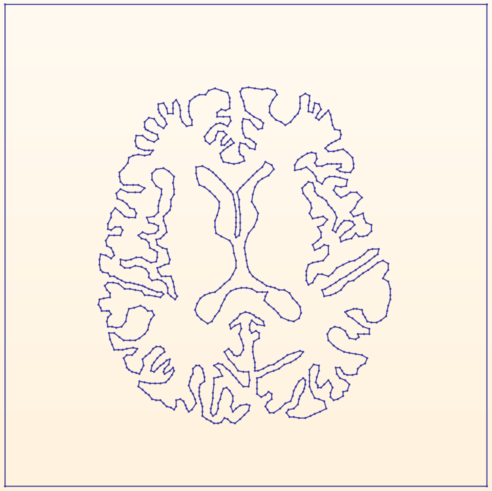
- Our algorithm can be extended in a straightforward manner to deal with arbitrary constrained triangulations.



- Given a constrained triangulation $\mathcal{T}$, build the dual graph $G$ of $\mathcal{T}$ respecting the constrained edges of $\mathcal{T}$. That is, the graph $G$ does not contain the dual edges of constrained edges of $\mathcal{T}$.
- Suppose that $G$ has $c$ connected components. Then, build the spanning forest $T = \{T_1, T_2, \ldots, T_c\}$ of $G$ and run our algorithm on each $T_i$.

# Constrained Quadrangulations

- Our algorithm can be extended in a straightforward manner to deal with arbitrary constrained triangulations.



- Given a constrained triangulation $\mathcal{T}$, build the dual graph $G$ of $\mathcal{T}$ respecting the constrained edges of $\mathcal{T}$. That is, the graph $G$ does not contain the dual edges of constrained edges of $\mathcal{T}$.
- Suppose that $G$ has $c$ connected components. Then, build the spanning forest $T = \{T_1, T_2, \ldots, T_c\}$ of $G$ and run our algorithm on each $T_i$.
- The number of quadrilaterals generated by the algorithm is at most $\lfloor \frac{3}{2}t \rfloor + 5c - 5$ and the number of Steiner points is at most $t + 4c - 3$.

# Implementation and Results

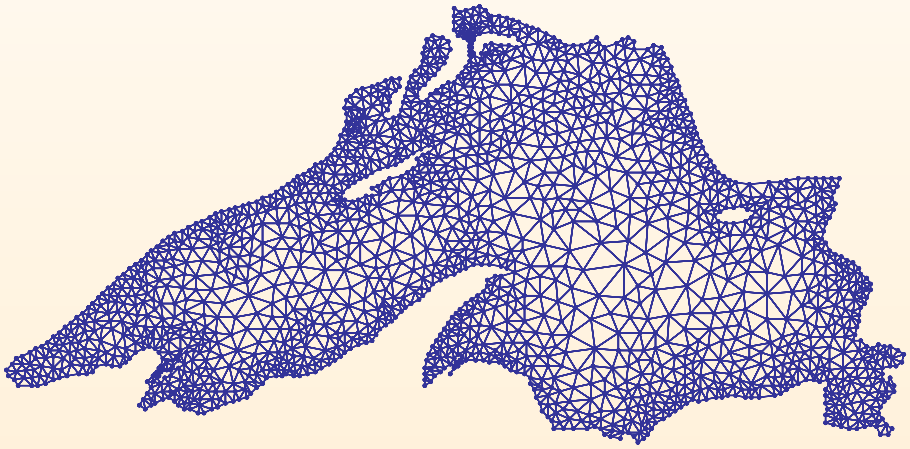
- We have an implementation of a slightly different version of this algorithm at

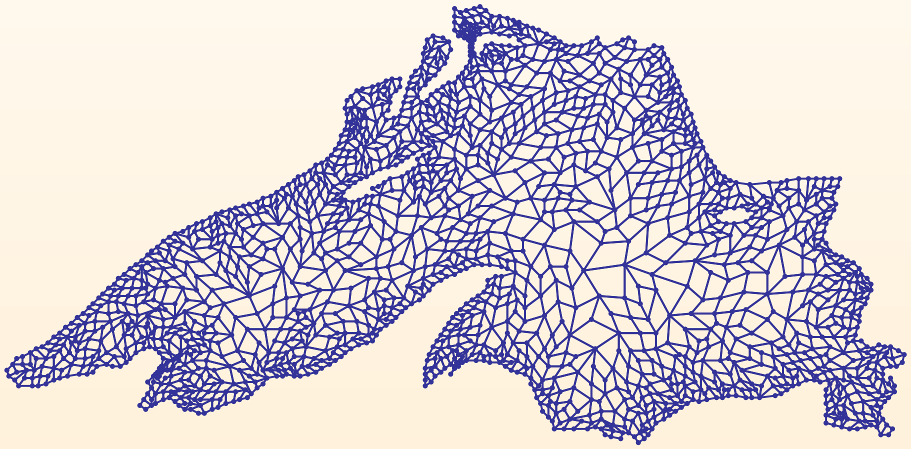
  *http://www.seas.upenn.edu/~marcelos/cqmesh.html*

# Implementation and Results

- We have an implementation of a slightly different version of this algorithm at

  *http://www.seas.upenn.edu/~marcelos/cqmesh.html*





- We have noticed that our algorithm generates about $0.6t$ quadrilaterals in most test cases, where $t$ is the number of triangles of the input triangulation.

- Our algorithm tends to preserve the input mesh grading.

- We have noticed that the better the shape of the input triangles is, the better the shape of the output quadrilaterals.

# Implementation and Results

- Our algorithm does not give any guarantee on the quality of the shape of mesh elements.

# Implementation and Results

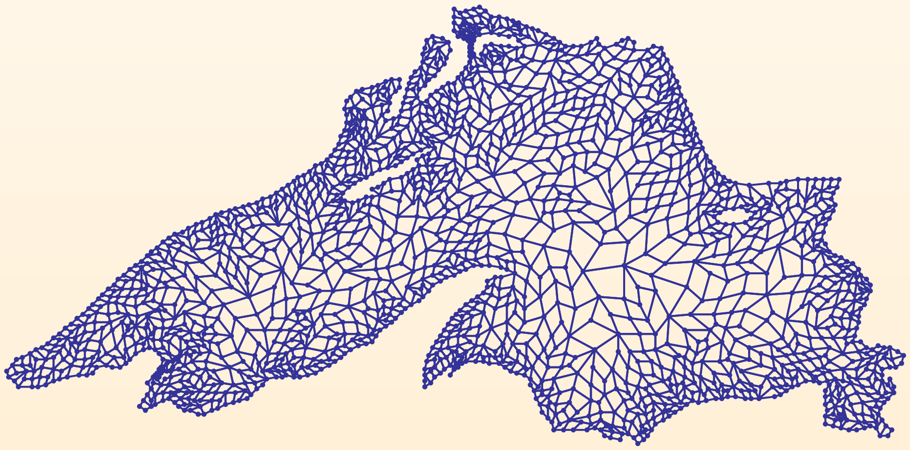
- Our algorithm does not give any guarantee on the quality of the shape of mesh elements.

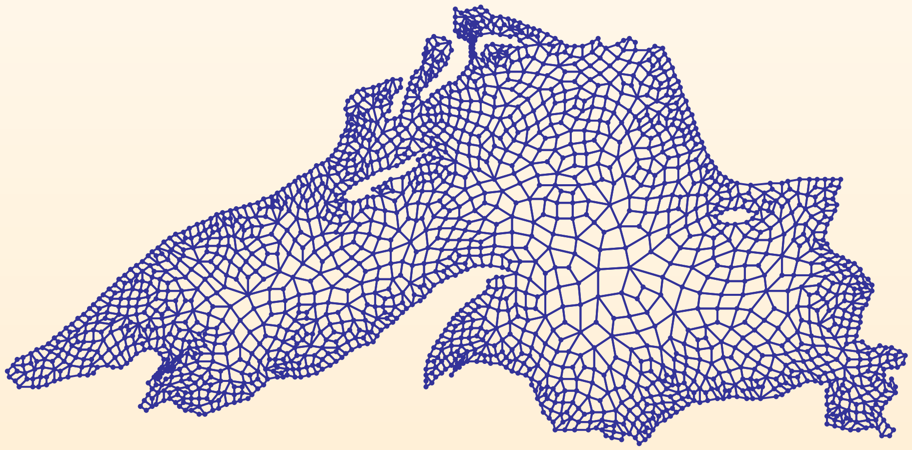
- Post-processing techniques (such as smoothing and topological improvements) should be used to increase mesh quality at the expenses of running time and mesh size.

# Implementation and Results

- Our algorithm does not give any guarantee on the quality of the shape of mesh elements.

- Post-processing techniques (such as smoothing and topological improvements) should be used to increase mesh quality at the expenses of running time and mesh size.



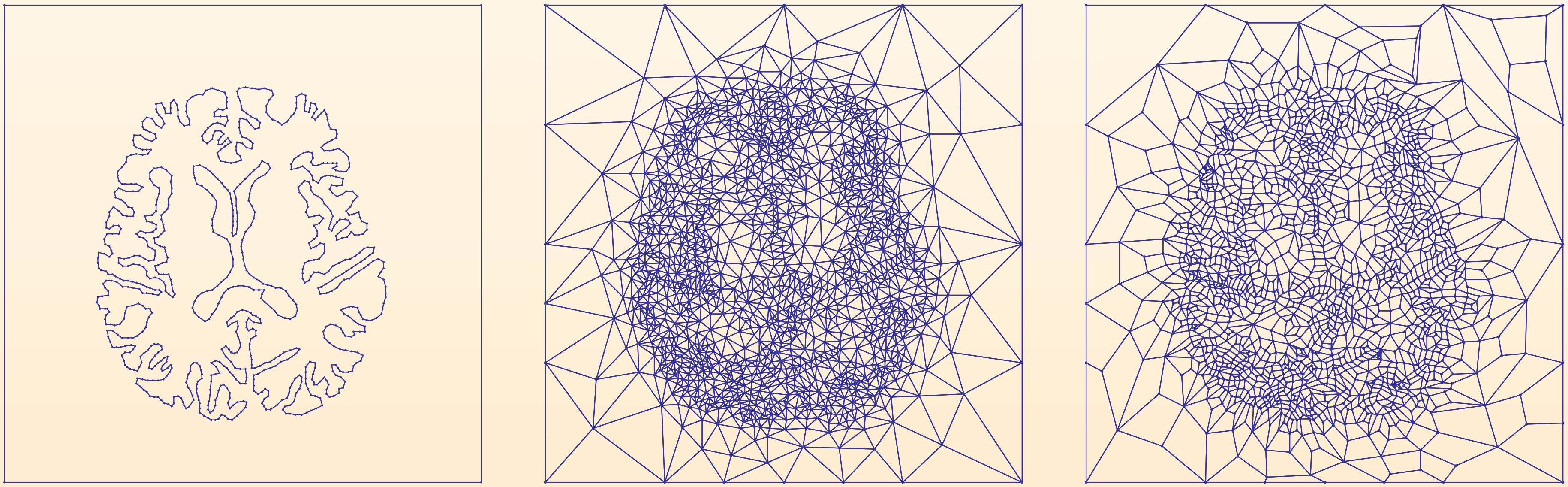


- The mesh on the right was obtained from the mesh on the left by using angle-based smoothing and topological clean-up.

- The mesh on the right has about $10\%$ more elements than the one on the left. The time to post-process the mesh on the right was 9 times longer than the one taken by our algorithm to generate the mesh on the left.

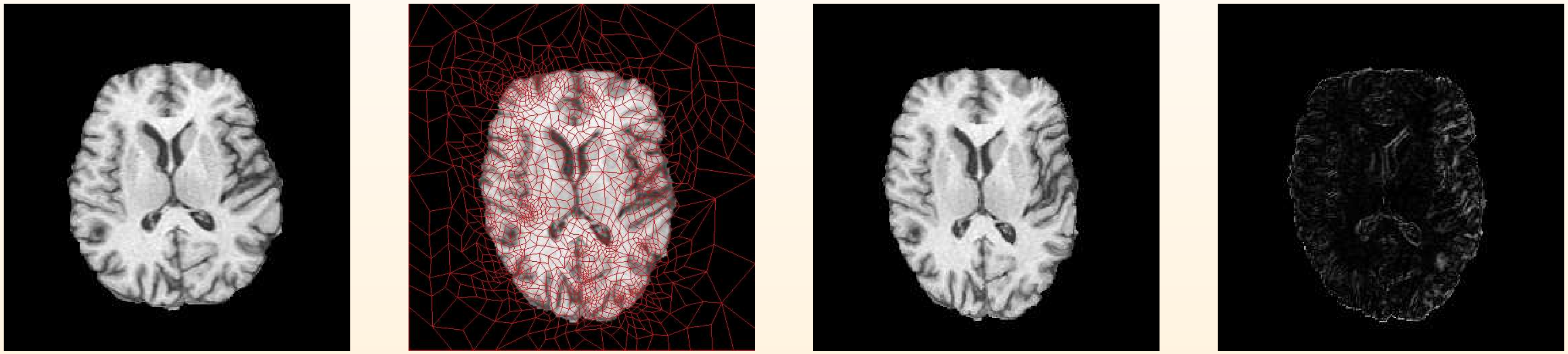
# Meshes from Images

- Image Segmentation
- Contour definition
- Polygonal approximation of contours



# Image registration

- The process of finding a spatial correspondence between two images.



- Elastic image registration (Broit, 1981)
- Variational formulation (Gee and Bajcsy, 1999)
- FE-based implementation of Gee and Bajcsy's registration method in the NLM Insight Segmentation and Registration Toolkit (ITK).

# Image Registration Experiment

- One pair $(A, B)$ of $256 \times 256$ pixel MR images.

# Image Registration Experiment

- One pair $(A, B)$ of $256 \times 256$ pixel MR images.

- Image $B$ is a deformation of image $A$ produced by a cubic polynomial.

# Image Registration Experiment

- One pair $(A, B)$ of $256 \times 256$ pixel MR images.

- Image $B$ is a deformation of image $A$ produced by a cubic polynomial.

- Image $B$ was partitioned using triangular meshes, quadrilateral grids, and quadrilateral meshes generated by our new algorithm.

# Image Registration Experiment

- One pair $(A, B)$ of $256 \times 256$ pixel MR images.

- Image $B$ is a deformation of image $A$ produced by a cubic polynomial.

- Image $B$ was partitioned using triangular meshes, quadrilateral grids, and quadrilateral meshes generated by our new algorithm.

- Triangular meshes produced by the software ‘‘Triangle’’ by Jonathan Shewchuk (`www-2.cs.cmu.edu/~quake/triangle.html`).

# Image Registration Experiment

- One pair $(A, B)$ of $256 \times 256$ pixel MR images.

- Image $B$ is a deformation of image $A$ produced by a cubic polynomial.

- Image $B$ was partitioned using triangular meshes, quadrilateral grids, and quadrilateral meshes generated by our new algorithm.

- Triangular meshes produced by the software "Triangle" by Jonathan Shewchuk (`www-2.cs.cmu.edu/~quake/triangle.html`).

- FEM-based image registration code from ITK.

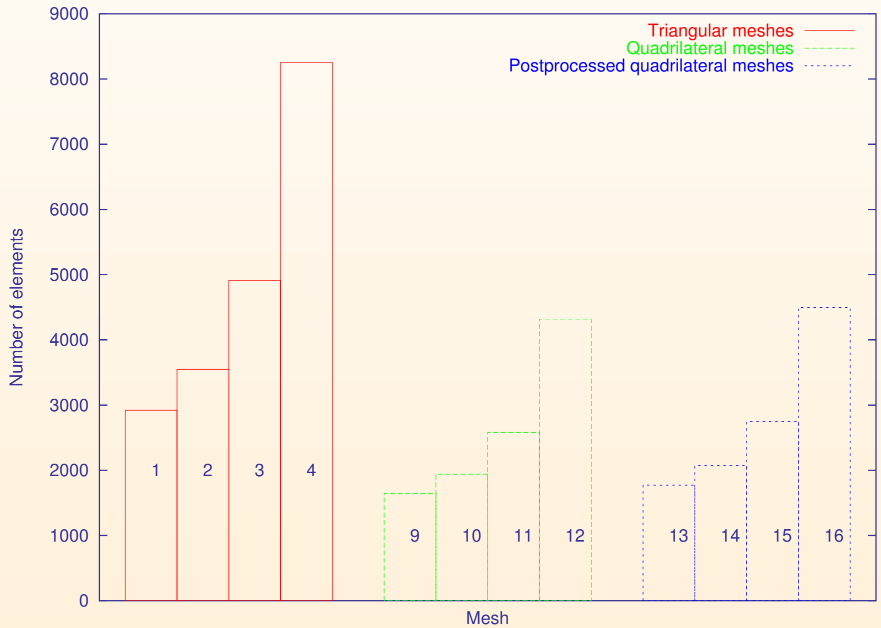
# Image Registration Experiment

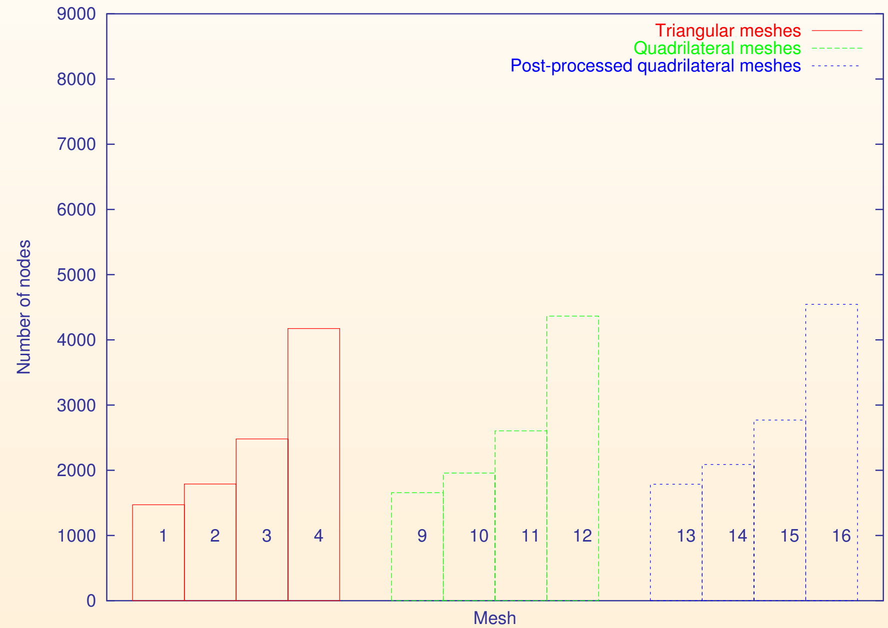
- One pair $(A, B)$ of $256 \times 256$ pixel MR images.
- Image $B$ is a deformation of image $A$ produced by a cubic polynomial.
- Image $B$ was partitioned using triangular meshes, quadrilateral grids, and quadrilateral meshes generated by our new algorithm.
- Triangular meshes produced by the software "Triangle" by Jonathan Shewchuk (`www-2.cs.cmu.edu/~quake/triangle.html`).
- FEM-based image registration code from ITK.
- We evaluate the results of the registration by calculating the RMS (root-mean squared) difference between the intensity values of corresponding pixels over the entire image domain.

# Performance Measurements

| Mesh | Description | #Elements | #Vertices |
|---|---|---|---|
| 1 | Triangular with minimum angle of 20 degrees | 2921 | 1472 |
| 2 | Triangular with minimum angle of 25 degrees | 3549 | 1790 |
| 3 | Triangular with minimum angle of 30 degrees | 4914 | 2481 |
| 4 | Triangular with minimum angle of 33 degrees | 8254 | 4173 |
| 5 | Quadrilateral grid of 8x8-pixel elements | 1024 | 1089 |
| 6 | Quadrilateral grid of 4x4-pixel elements | 4096 | 4225 |
| 7 | Quadrilateral grid of 2x2-pixel elements | 16384 | 16641 |
| 8 | Quadrilateral grid of 1x1-pixel elements | 65536 | 66049 |
| 9 | Quadrilateral mesh from triangular mesh 1 | 1645 | 1654 |
| 10 | Quadrilateral mesh from triangular mesh 2 | 1941 | 1957 |
| 11 | Quadrilateral mesh from triangular mesh 3 | 2581 | 2605 |
| 12 | Quadrilateral mesh from triangular mesh 4 | 4318 | 4364 |
| 13 | Quadrilateral mesh 9 after post-processing | 1773 | 1785 |
| 14 | Quadrilateral mesh 10 after post-processing | 2073 | 2089 |
| 15 | Quadrilateral mesh 11 after post-processing | 2747 | 2771 |
| 16 | Quadrilateral mesh 12 after post-processing | 4499 | 4545 |

# Performance Measurements



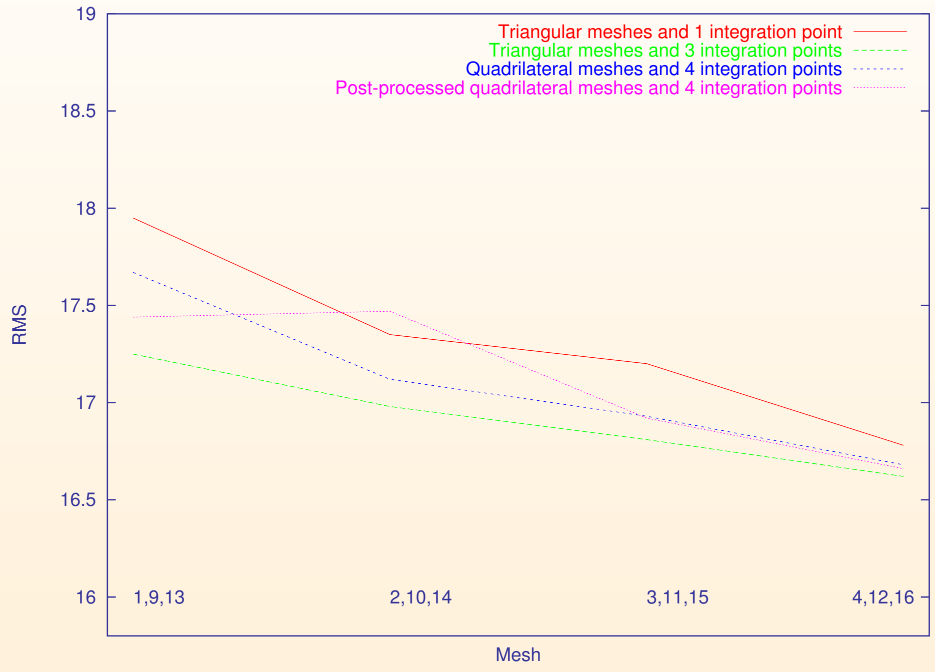




- Number of elements in the brain meshes generated by our algorithm is about $60\%$ of the number of elements in their triangular counterparts.
- Number of vertices in the brain meshes generated by our algorithm is slightly bigger than the number of vertices in their triangular counterparts.

# Performance Measurements

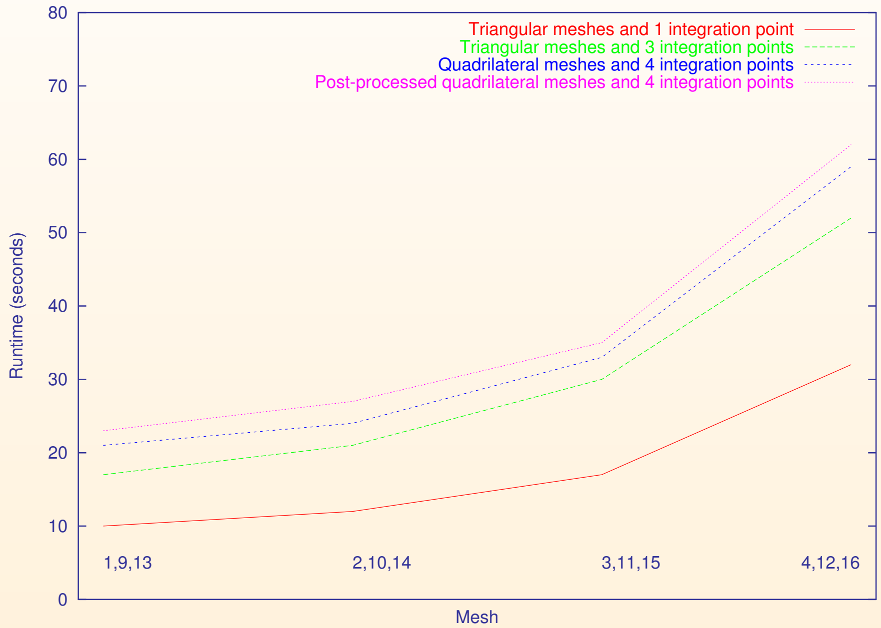
| Mesh | #Elements | Int. Pts. | Runtime (sec.) | RMS |
|---|---|---|---|---|
| 1 | 2921 | 1 | 10 | 17.95 |
| 1 | 2991 | 3 | 17 | 17.25 |
| 2 | 3549 | 1 | 12 | 17.35 |
| 2 | 3549 | 3 | 21 | 16.98 |
| 3 | 4914 | 1 | 17 | 17.20 |
| 3 | 4914 | 3 | 30 | 16.81 |
| 4 | 8254 | 1 | 32 | 16.78 |
| 4 | 8254 | 3 | 52 | 16.62 |
| 5 | 1024 | 4 | 12 | 18.56 |
| 6 | 4096 | 4 | 46 | 16.99 |
| 7 | 16384 | 4 | 205 | 16.11 |
| 8 | 65536 | 4 | 1001 | 15.93 |
| 9 | 1645 | 4 | 21 | 17.67 |
| 10 | 1941 | 4 | 24 | 17.12 |
| 11 | 2581 | 4 | 33 | 16.93 |
| 12 | 4318 | 4 | 59 | 16.68 |
| 13 | 1773 | 4 | 23 | 17.44 |
| 14 | 2073 | 4 | 27 | 17.47 |
| 15 | 2747 | 4 | 35 | 16.92 |
| 16 | 4499 | 4 | 62 | 16.66 |

# Performance Measurements




- Even though the number of elements of the quadrilaterals meshes generated by our algorithm is about $60\%$ of the number of elements in their triangular counterparts, the RMS's due to the quadrilateral meshes are comparable with the ones of their triangular counterparts.

# Performance Measurements




- Runtime associated with the quadrilateral meshes are larger than the ones associated with the triangular meshes.
- Brain meshes $11$ and $15$, which are generated by our algorithm, have less than $\frac{3}{4}$ of the number of elements of mesh $6$ (a uniform grid) and yet they both have a smaller RMS associated with them.

# Conclusions

- We developed an algorithm to generate quadrilateral meshes of arbitrary polygonal regions with or without holes.

# Conclusions

- We developed an algorithm to generate quadrilateral meshes of arbitrary polygonal regions with or without holes.

- Our algorithm is provably guaranteed to generate quadrangulateral meshes of bounded size and its bounds are better than the ones provided by similar algorithms using the indirect approach.

# Conclusions

- We developed an algorithm to generate quadrilateral meshes of arbitrary polygonal regions with or without holes.

- Our algorithm is provably guaranteed to generate quadrangulateral meshes of bounded size and its bounds are better than the ones provided by similar algorithms using the indirect approach.

- Our algorithm is simpler and faster than most of the algorithms for generating quadrilateral meshes with high quality element shape, and it can be a reasonable choice if element shape quality is not an issue.

# Conclusions

- We developed an algorithm to generate quadrilateral meshes of arbitrary polygonal regions with or without holes.

- Our algorithm is provably guaranteed to generate quadrangulateral meshes of bounded size and its bounds are better than the ones provided by similar algorithms using the indirect approach.

- Our algorithm is simpler and faster than most of the algorithms for generating quadrilateral meshes with high quality element shape, and it can be a reasonable choice if element shape quality is not an issue.

- We applied our algorithm to generate quadrilateral meshes from imaging data, and then evaluate mesh quality with respect to the performance of a FE-based image registration method implemented in ITK.

# Conclusions

- We developed an algorithm to generate quadrilateral meshes of arbitrary polygonal regions with or without holes.

- Our algorithm is provably guaranteed to generate quadrangulateral meshes of bounded size and its bounds are better than the ones provided by similar algorithms using the indirect approach.

- Our algorithm is simpler and faster than most of the algorithms for generating quadrilateral meshes with high quality element shape, and it can be a reasonable choice if element shape quality is not an issue.

- We applied our algorithm to generate quadrilateral meshes from imaging data, and then evaluate mesh quality with respect to the performance of a FE-based image registration method implemented in ITK.

- Our evaluation has shown that our meshes are comparable with their triangular counterparts, and they are slightly better than the uniform grids automatically provided by ITK.

# Future Work

- We would like to investigate and formalize the relationship between the quality of input triangulation (in terms of angle bounds, for instance) and the quality of the quadrilateral mesh produced by our algorithm.

# Future Work

- We would like to investigate and formalize the relationship between the quality of input triangulation (in terms of angle bounds, for instance) and the quality of the quadrilateral mesh produced by our algorithm.

- We also intend to extend our image registration experiment to include an advancing front method that generates quadrilateral meshes.

# Future Work

- We would like to investigate and formalize the relationship between the quality of input triangulation (in terms of angle bounds, for instance) and the quality of the quadrilateral mesh produced by our algorithm.

- We also intend to extend our image registration experiment to include an advancing front method that generates quadrilateral meshes.

- Finally, we would like to extend our algorithm to build three-dimensional quadrilateral meshes from imaging data. One possibility is to start with a ''reconstruction from slices'' approach.