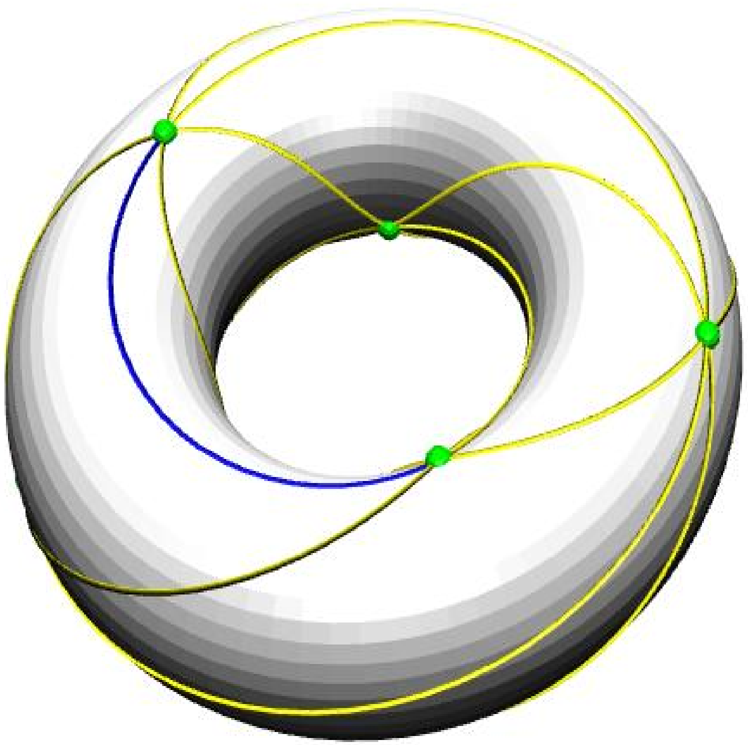
Visualizing Graphs on the Torus

Robert Löwe

Richard Sieg



Content

1. Motivation and Mathematical Background
   (a) Heawood's Bound and the K_7 on the Torus
   (b) The Forced-Based Algorithm
2. Implementation
   (a) Data Structure for Graphs on the Torus
   (b) Implementation of the Forced-Based Algorithm
   (c) The Project
3. The User Interface
4. Outlook

Heawood's Bound

Theorem [Ringel & Youngs, 1968]

For a manifold M that is not S^2 or the Klein bottle, the following are equivalent:

(i) There is an embedding $K_n \hookrightarrow M$

(ii) $n \leq \frac{1}{2}(7 + \sqrt{49 - 24\chi(M)})$,

where K_n denotes the complete graph with n vertices and $\chi(M)$ the Euler characteristic.

Heawood's Bound

Theorem [Ringel & Youngs, 1968]

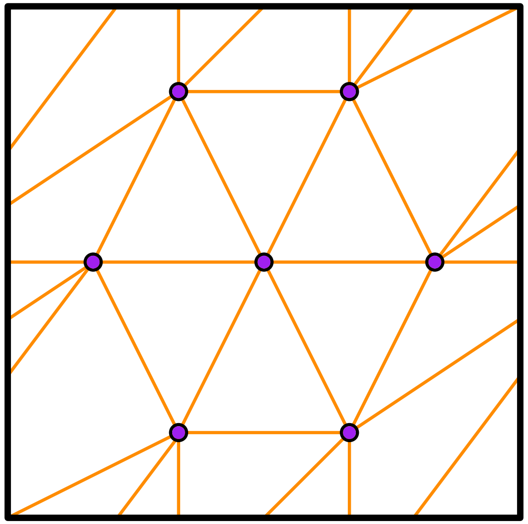
For a manifold M that is not S^2 or the Klein bottle, the following are equivalent:

(i) There is an embedding $K_n \hookrightarrow M$

(ii) $n \leq \frac{1}{2}(7 + \sqrt{49 - 24\chi(M)})$,

where K_n denotes the complete graph with n vertices and $\chi(M)$ the Euler characteristic.

Since $\chi(T^2) = 0$ it is possible to embed the K_7 on the Torus.



Heawood's Bound

Motivation:
A nice picture of the K_7 embedded on the torus.

Heawood's Bound

Motivation:
A nice picture of the K_7 embedded on the torus.

↓

Goal:
Find an appropriate data structure that designs arbitrary graphs on the torus and an algorithm to "beautify" them.

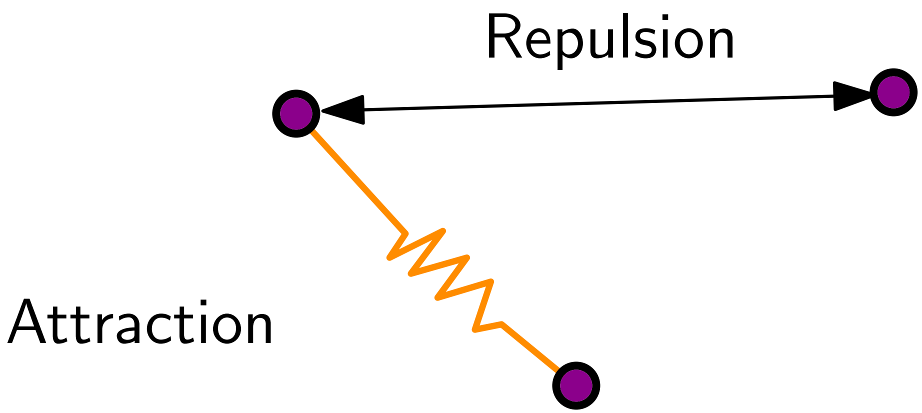
The Force-Based Algorithm

Desire:
Move the vertices of a graph, such that they have a more or less equal distance and their egdes the same length (*equilibrium*).

The Force-Based Algorithm

Desire:
Move the vertices of a graph, such that they have a more or less equal distance and their egdes the same length (*equilibrium*).

Idea:
Consider the vertices as *electrons* that push off each other and edges as springs.



The Force-Based Algorithm

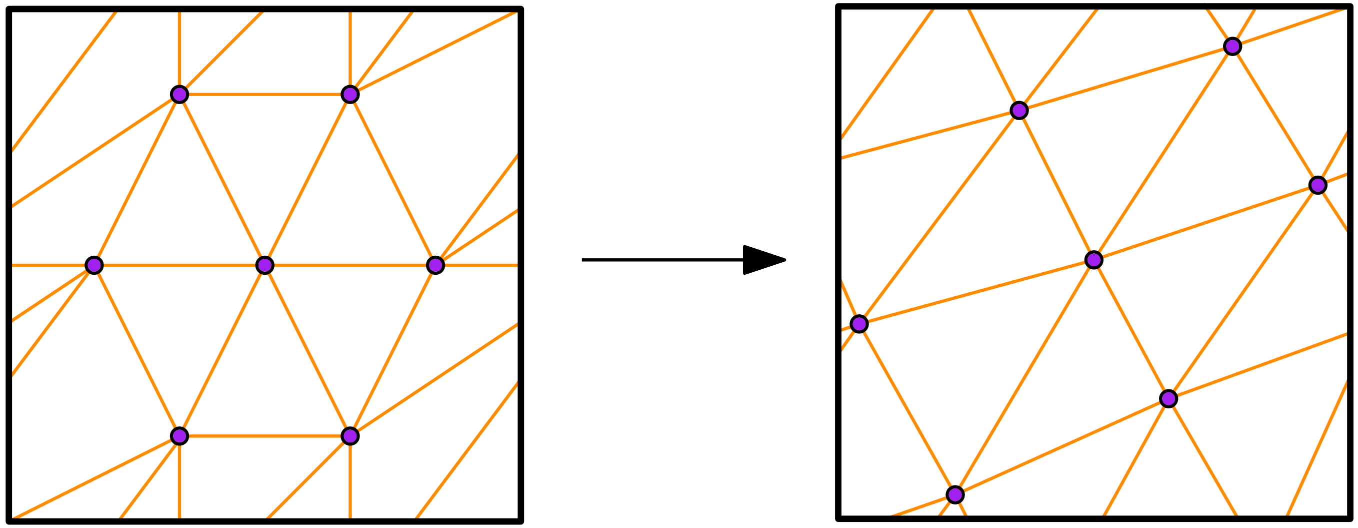
Repuslion is not necessary for non-trivial cases on the torus.

Result:

The Force-Based Algorithm

Repuslion is not necessary for non-trivial cases on the torus.

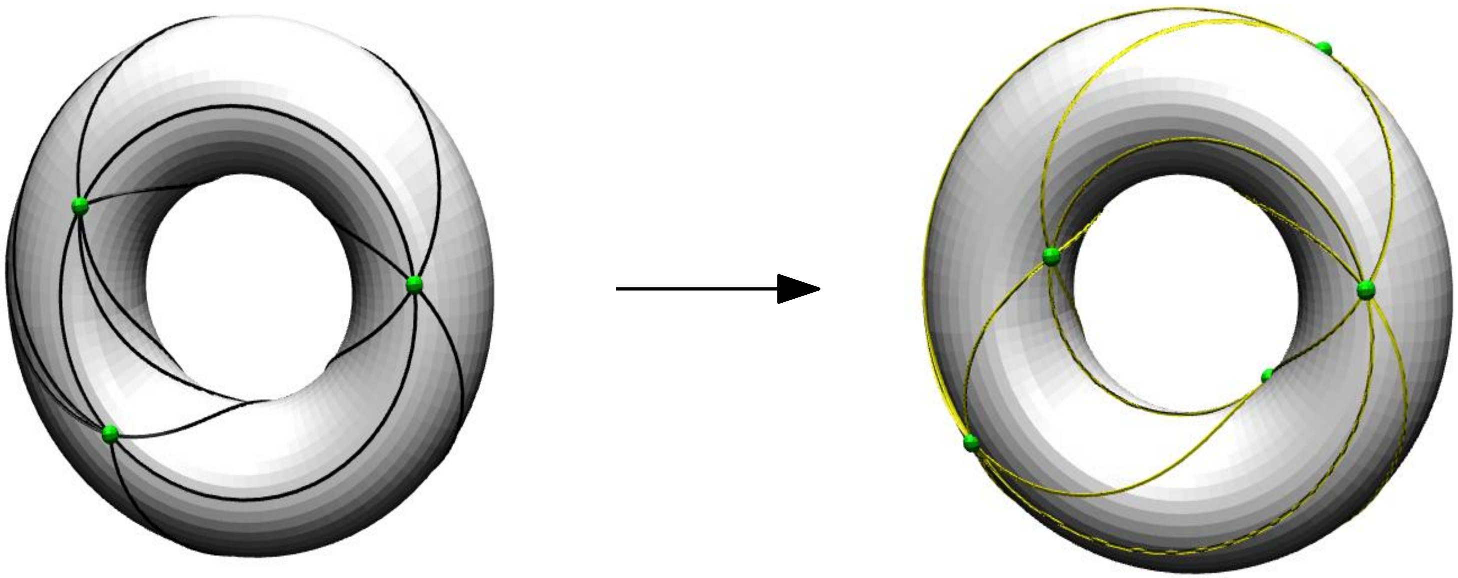
Result:



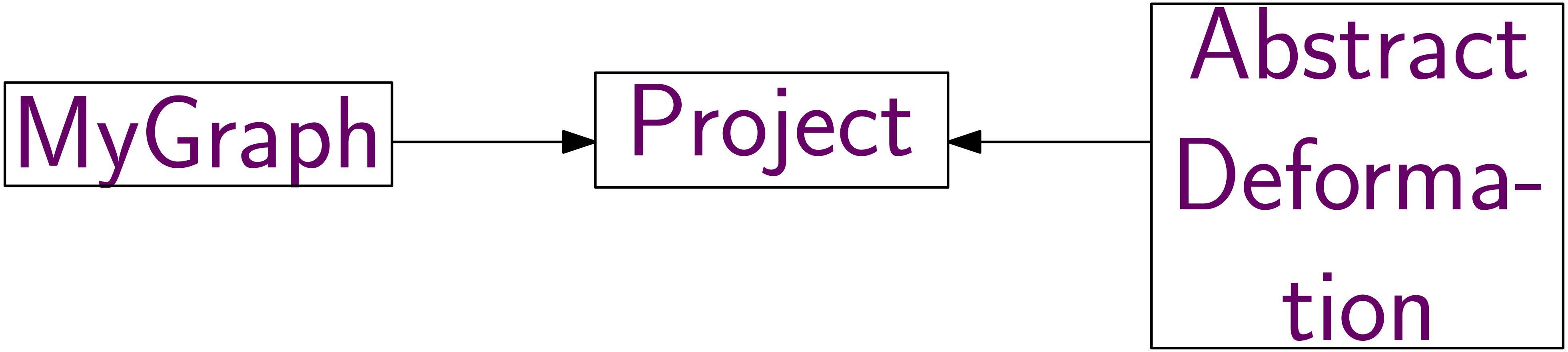
The Force-Based Algorithm

Repuslion is not necessary for non-trivial cases on the torus.

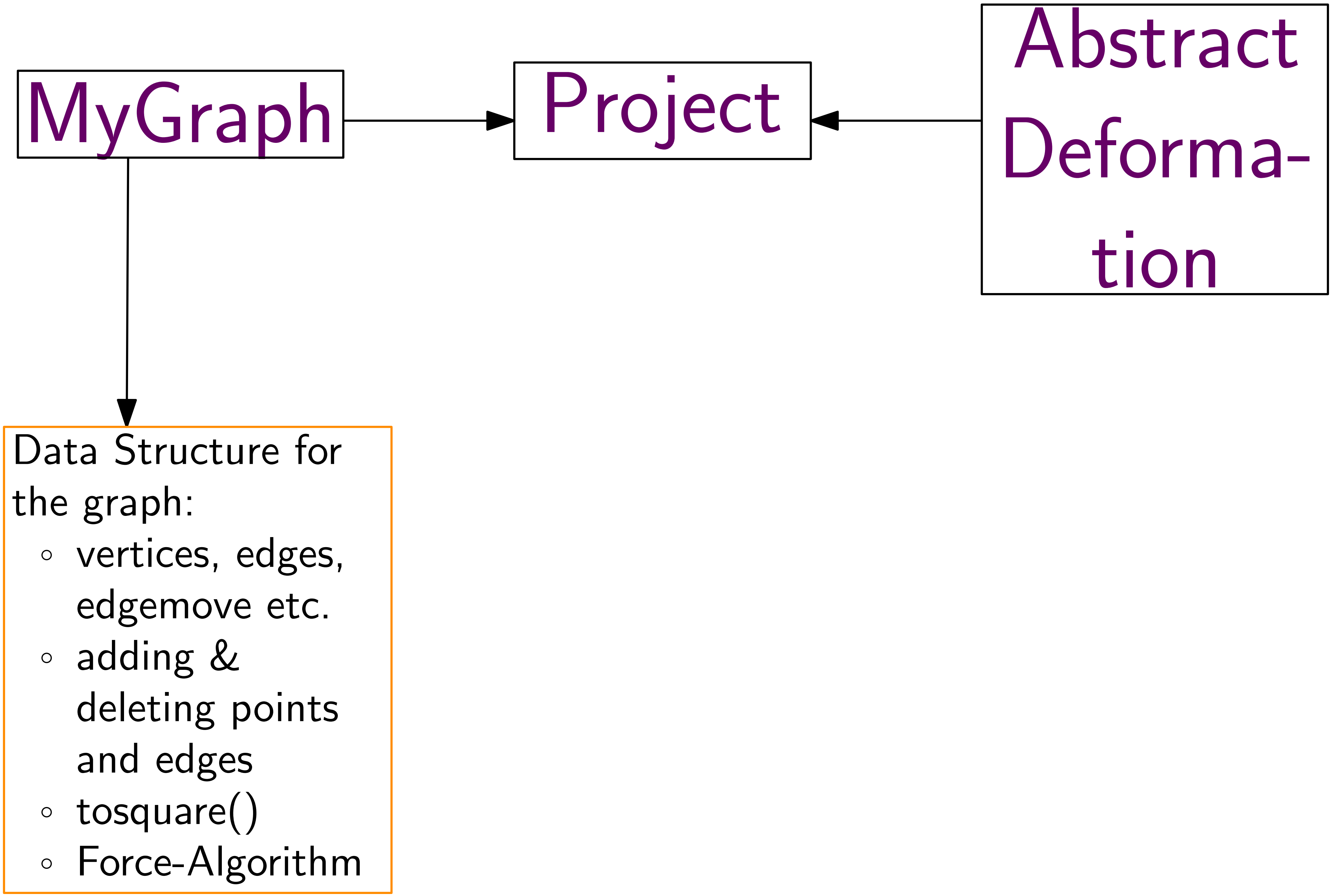
Result:



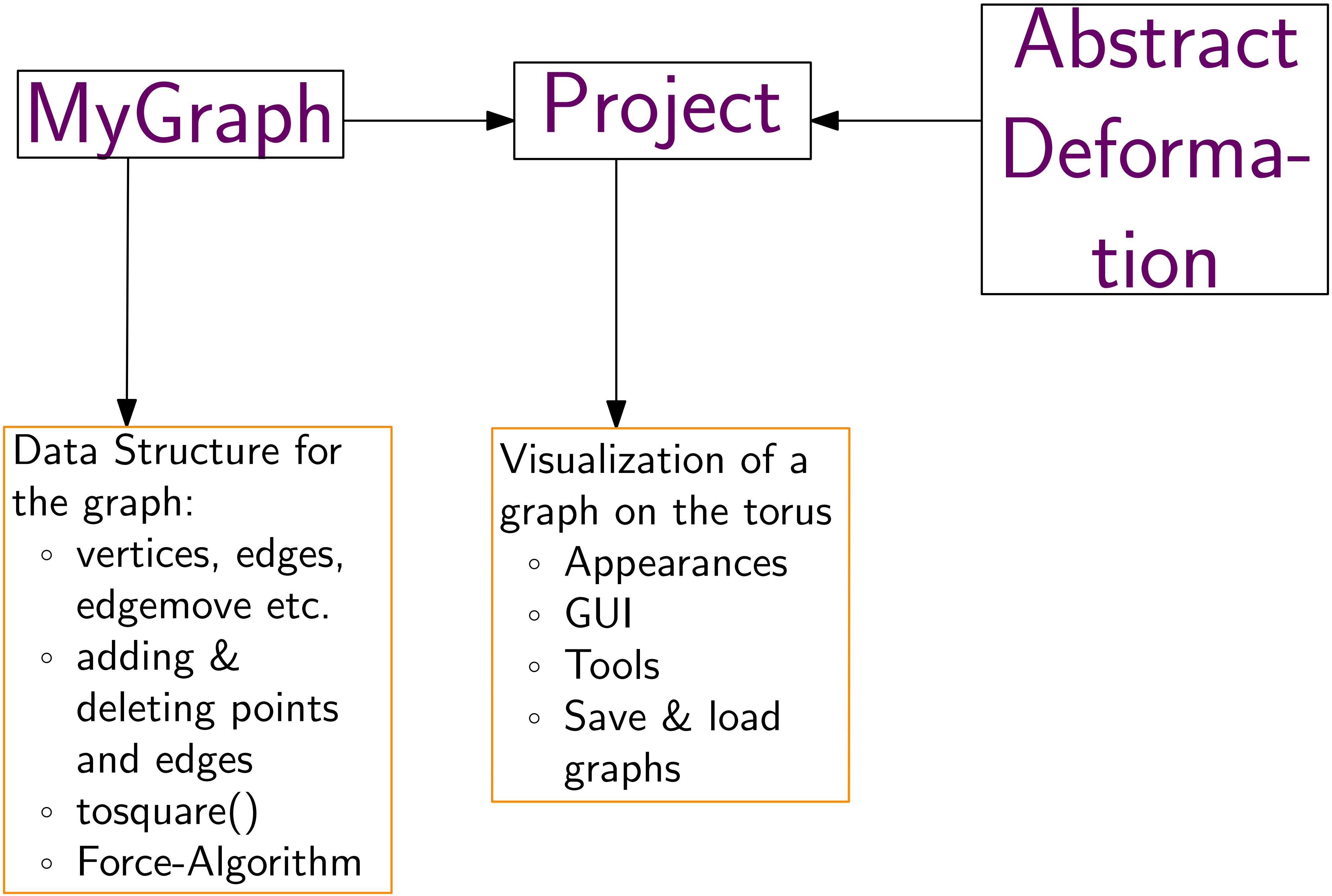
Data Structure



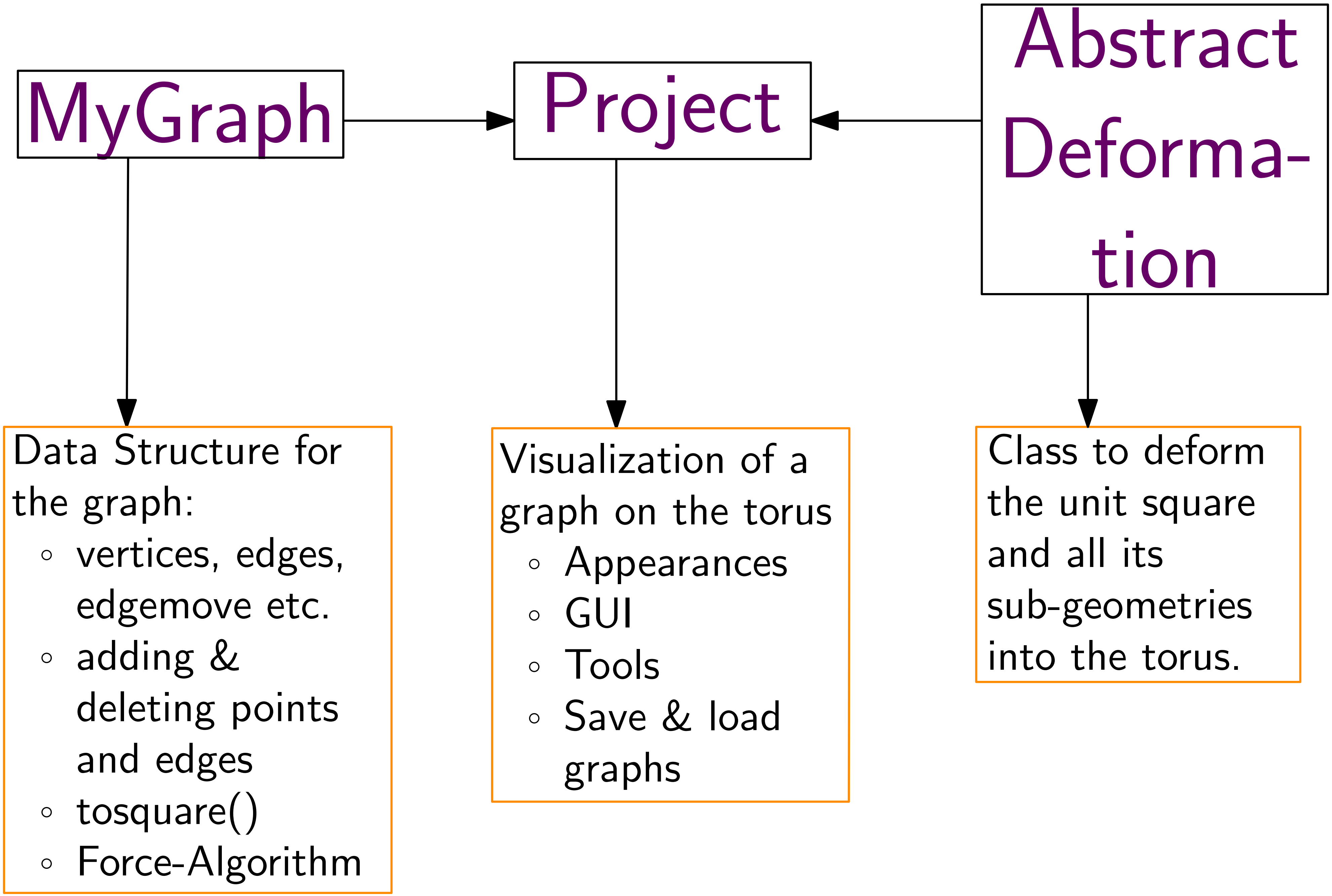
Data Structure



Data Structure



Data Structure



MyGraph

Fields:

- `double[] vertices`
- `byte[][] edges`
- `int[][][] edge_move`: encodes every edge with a translation (m, n)
- fields for the physical constants and active vertices etc.

MyGraph

Fields:

- `double[] vertices`
- `byte[][] edges`
- `int[][][] edge_move`: encodes every edge with a translation (m, n)
- fields for the physical constants and active vertices etc.

Methods:

- three different constructors
- all necessary getter and setter
- add and delete points and edges
- `tosquare()`: creates the final IndexedLineSet
- `theForce(double a, double b, double c)`: a single step of the Force-Algorithm

MyGraph

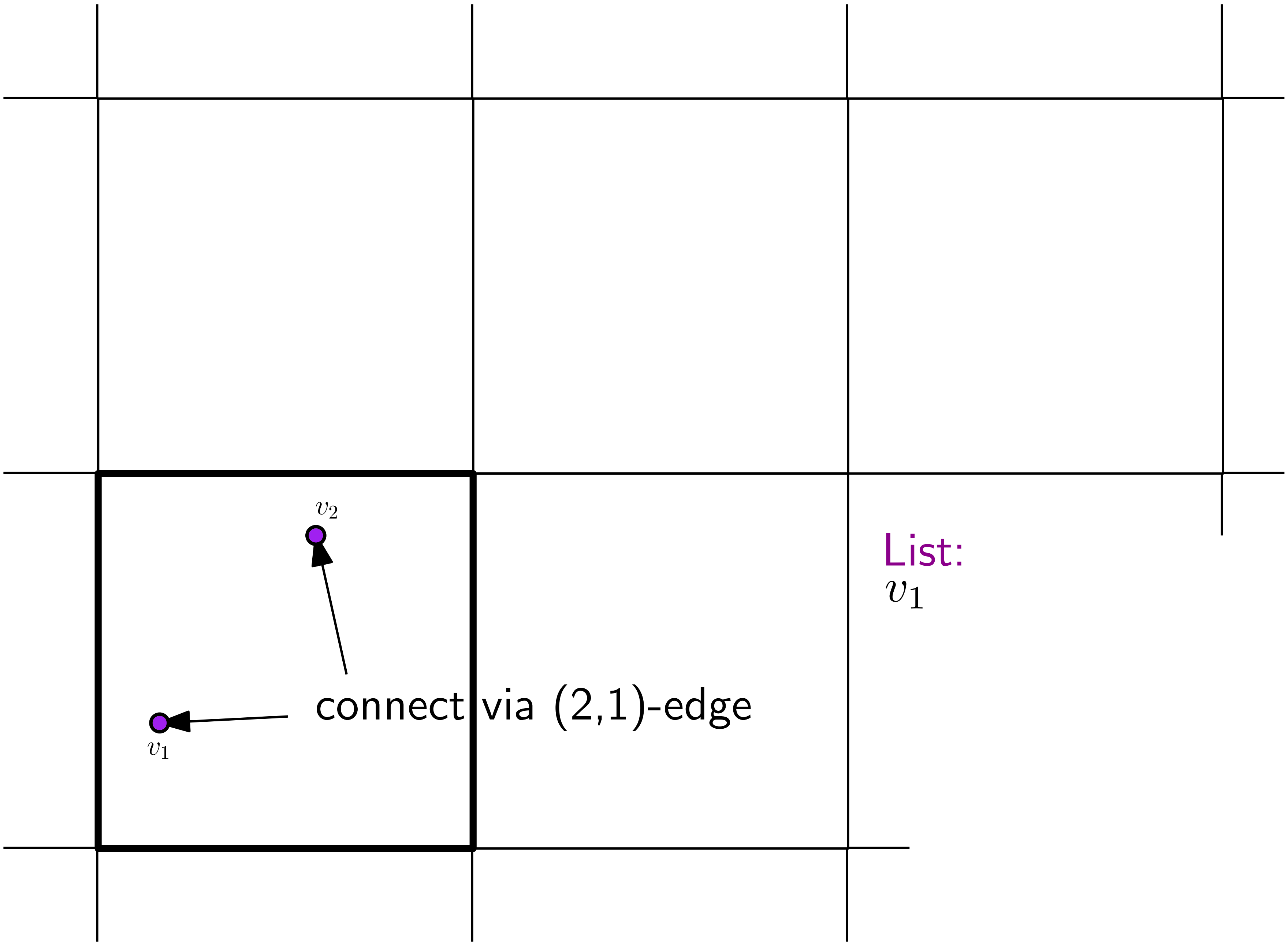
Fields:

- `double[] vertices`
- `byte[][] edges`
- `int[][][] edge_move`: encodes every edge with a translation (m, n)
- fields for the physical constants and active vertices etc.

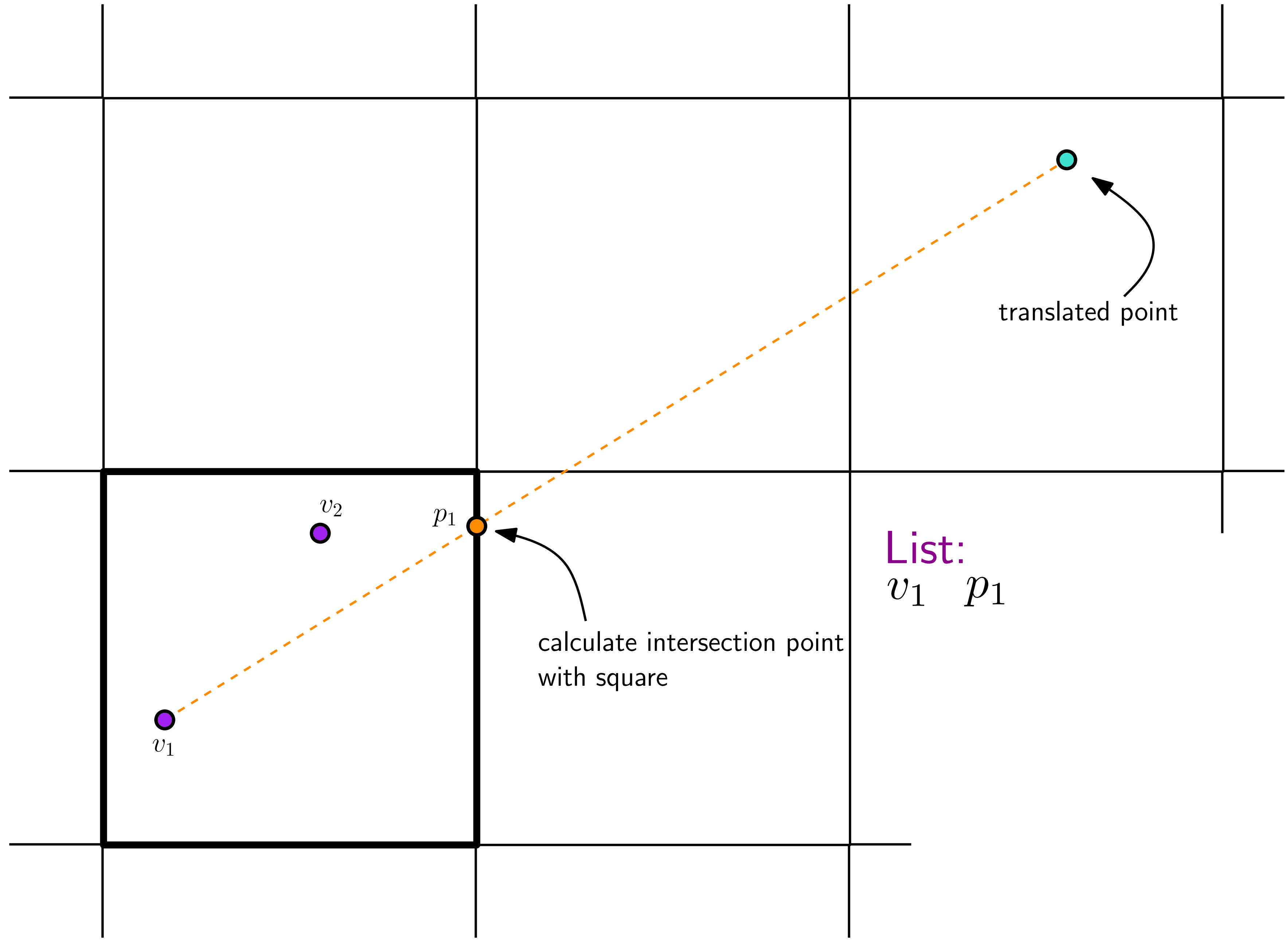
Methods:

- three different constructors
- all necessary getter and setter
- add and delete points and edges
- `tosquare()`: creates the final IndexedLineSet
- `theForce(double a, double b, double c)`: a single step of the Force-Algorithm

tosquare()

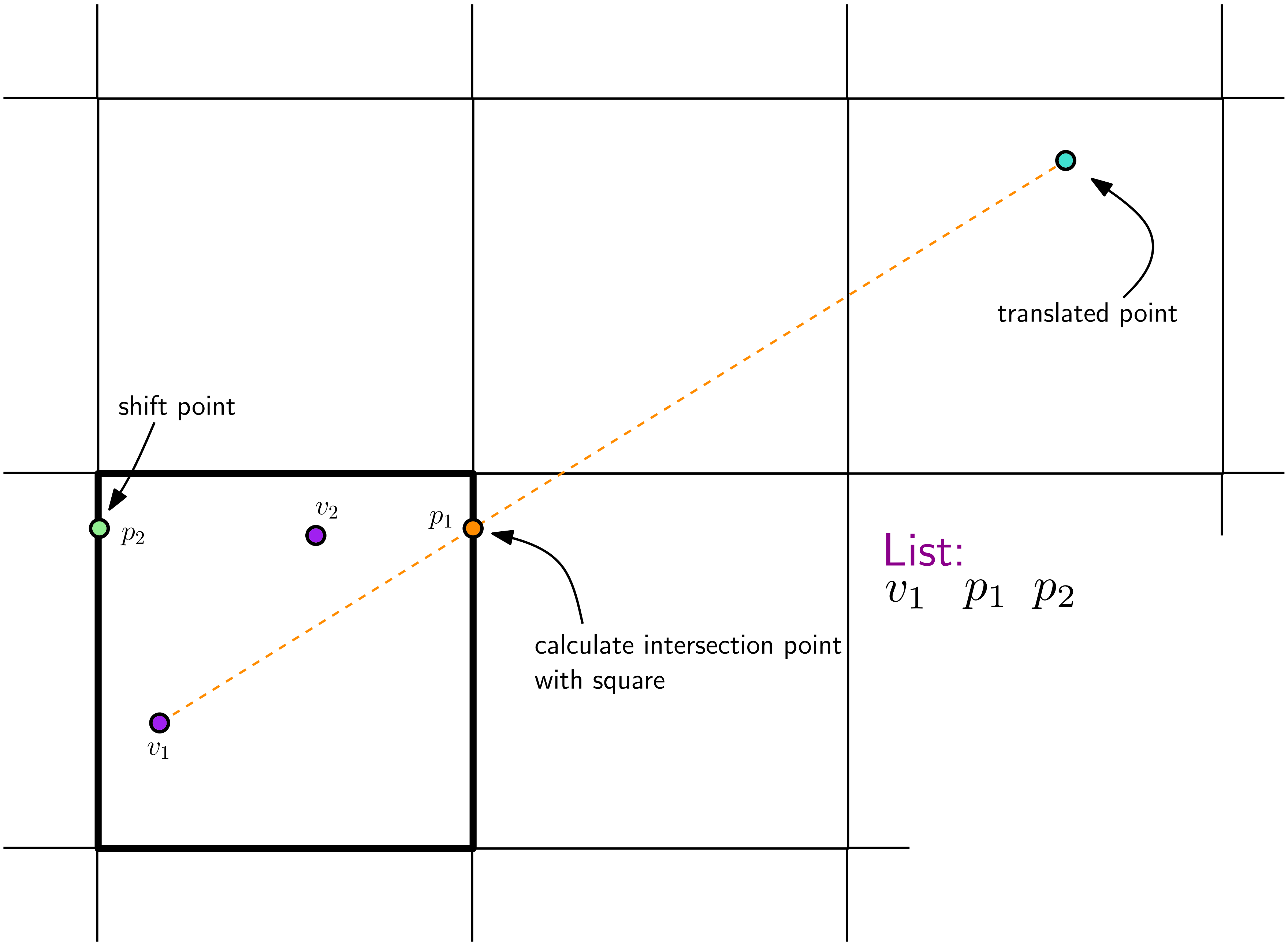


tosquare()



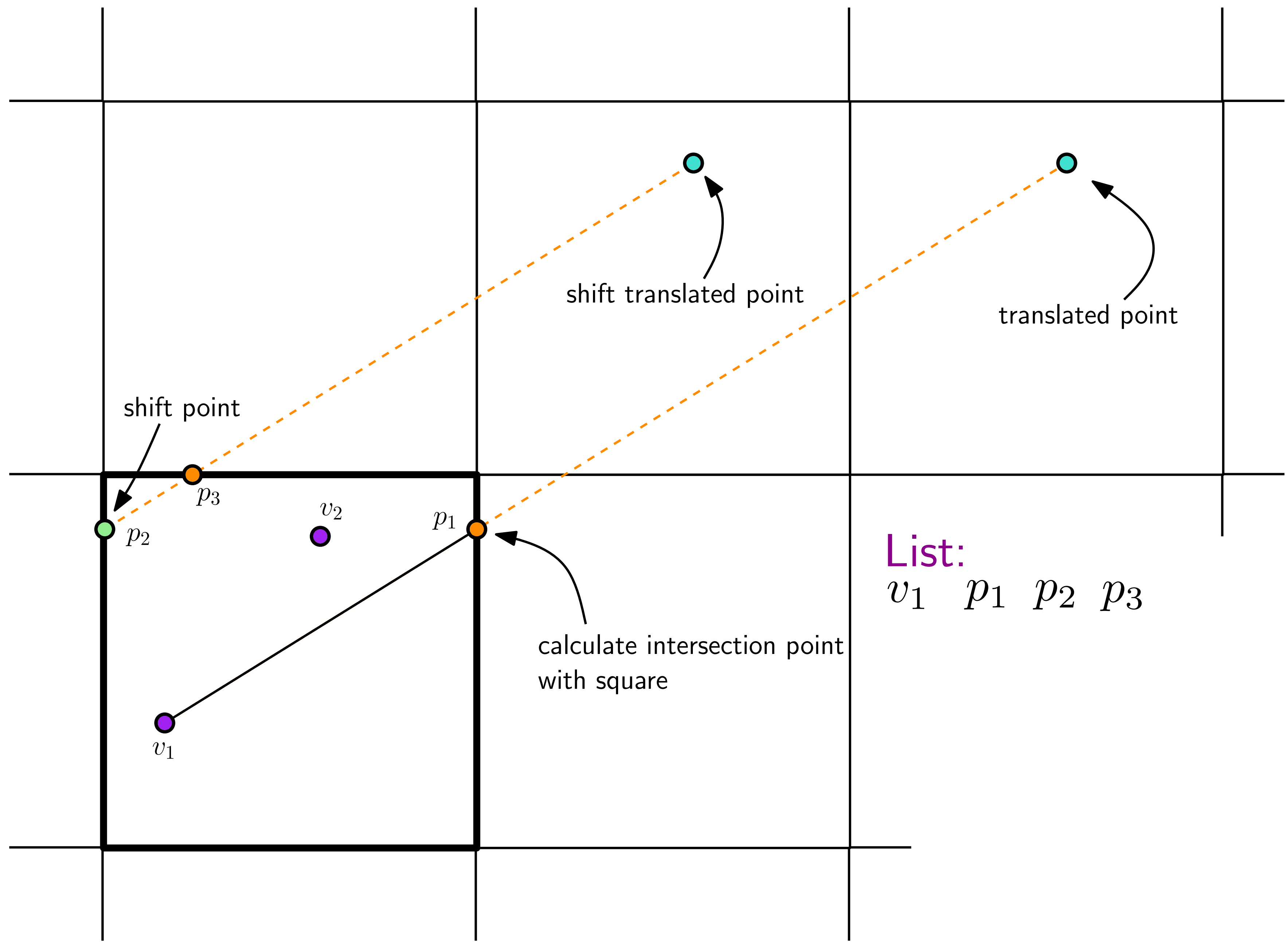
v_2
p_1
v_1
translated point
calculate intersection point
with square
List:
v_1 p_1

tosquare()

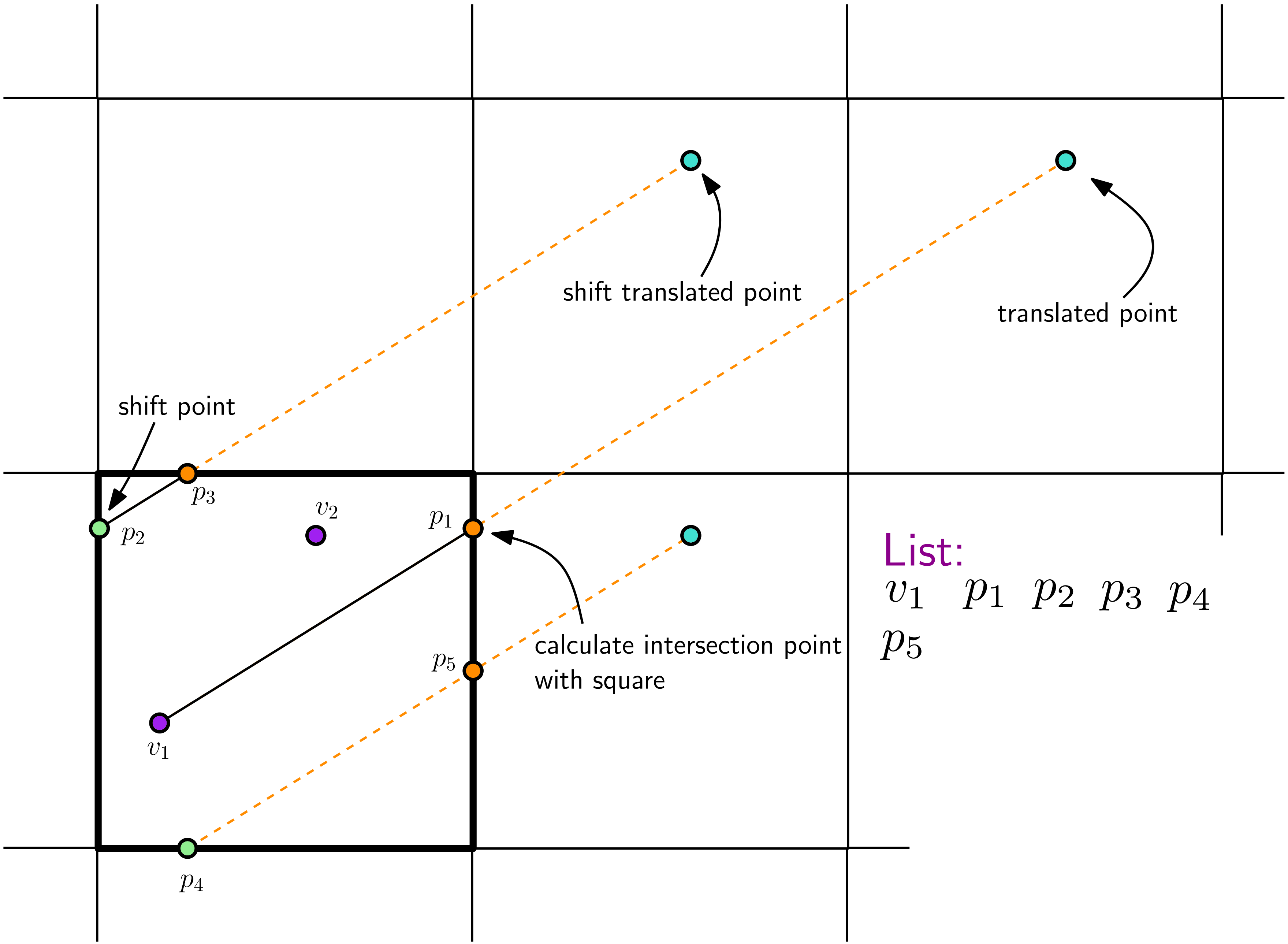


translated point
shift point
v_2
p_1
p_2
List:
v_1 p_1 p_2
calculate intersection point
with square
v_1

tosquare()


shift translated point
translated point
shift point
p_3
v_2
p_1
p_2
calculate intersection point
with square
v_1
List:
v_1 p_1 p_2 p_3

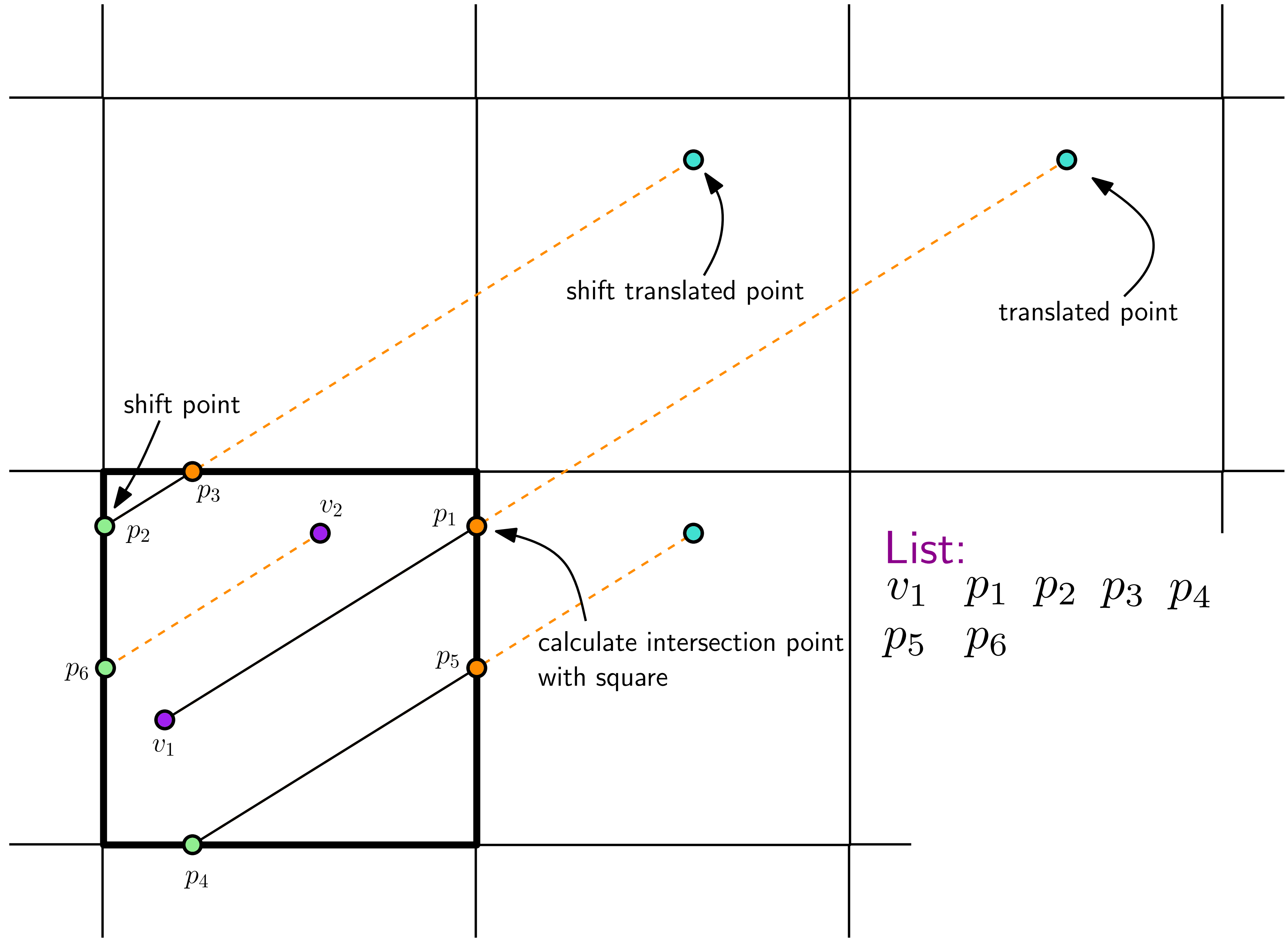
tosquare()



List:

v_1 p_1 p_2 p_3 p_4 p_5

tosquare()

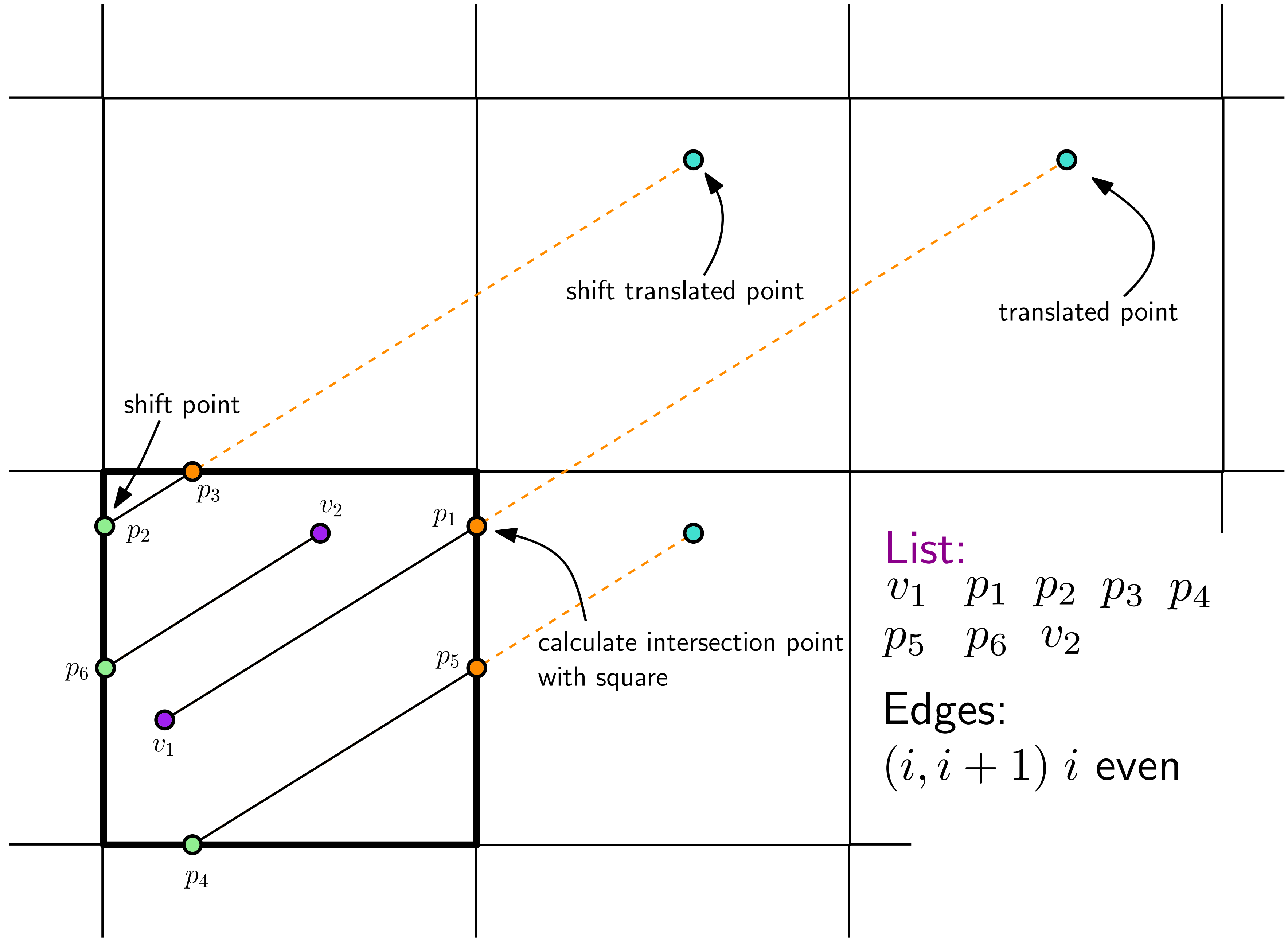


List:

v_1 p_1 p_2 p_3 p_4

p_5 p_6

tosquare()



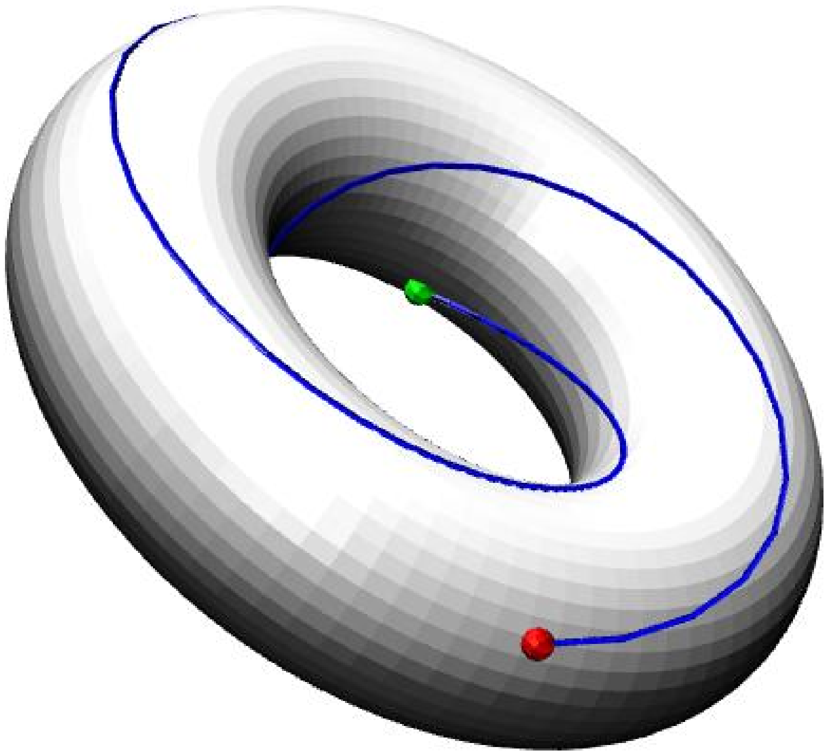
List:

v_1 p_1 p_2 p_3 p_4 p_5 p_6 v_2

Edges:

$(i, i+1)$ i even

tosquare()



The Force-Based Algorithm

if equilibrium = true **then return**
end if
for all $v_i \in V$ **do** ▷ Calculate repulsion
  $net(i) \leftarrow (0, 0)$
  for all $v_j \in V, v_j \neq v_i$ **do**
    calculate closest representative $\tilde{v}_j$ of v_j to v_i
    $net(i) \leftarrow net(i) + k_1 \cdot (v_i - \tilde{v}_j)/\|v_i - \tilde{v}_j\|^2$
  end for
  for all $v_j \in V$ **do** ▷ Calculate Attraction
    $\tilde{v}_j \leftarrow v_j + edgemove(v_i, v_j)$ ▷ translate v_j
    if $(v_i, v_j) \in E$ **then**
      $net(i) \leftarrow net(i) + k_2 \cdot (\tilde{v}_j - v_i)$
    end if
  end for
  $velo(i) \leftarrow k_3 \cdot (velo(i) + net(i))$
end for

The Force-Based Algorithm

```
for all  v_i ∈ V do
    if ||velo(i)|| > ε then break
    end if
    if at last vertex then
        equilibrium=true
        return                          ▷ Reached equilibrium
    end if
end for
for all  v_i ∈ V do
    v_i ← v_i + velo(i)
    ▷ in consideration of the boundarys of the flat torus
end for
```

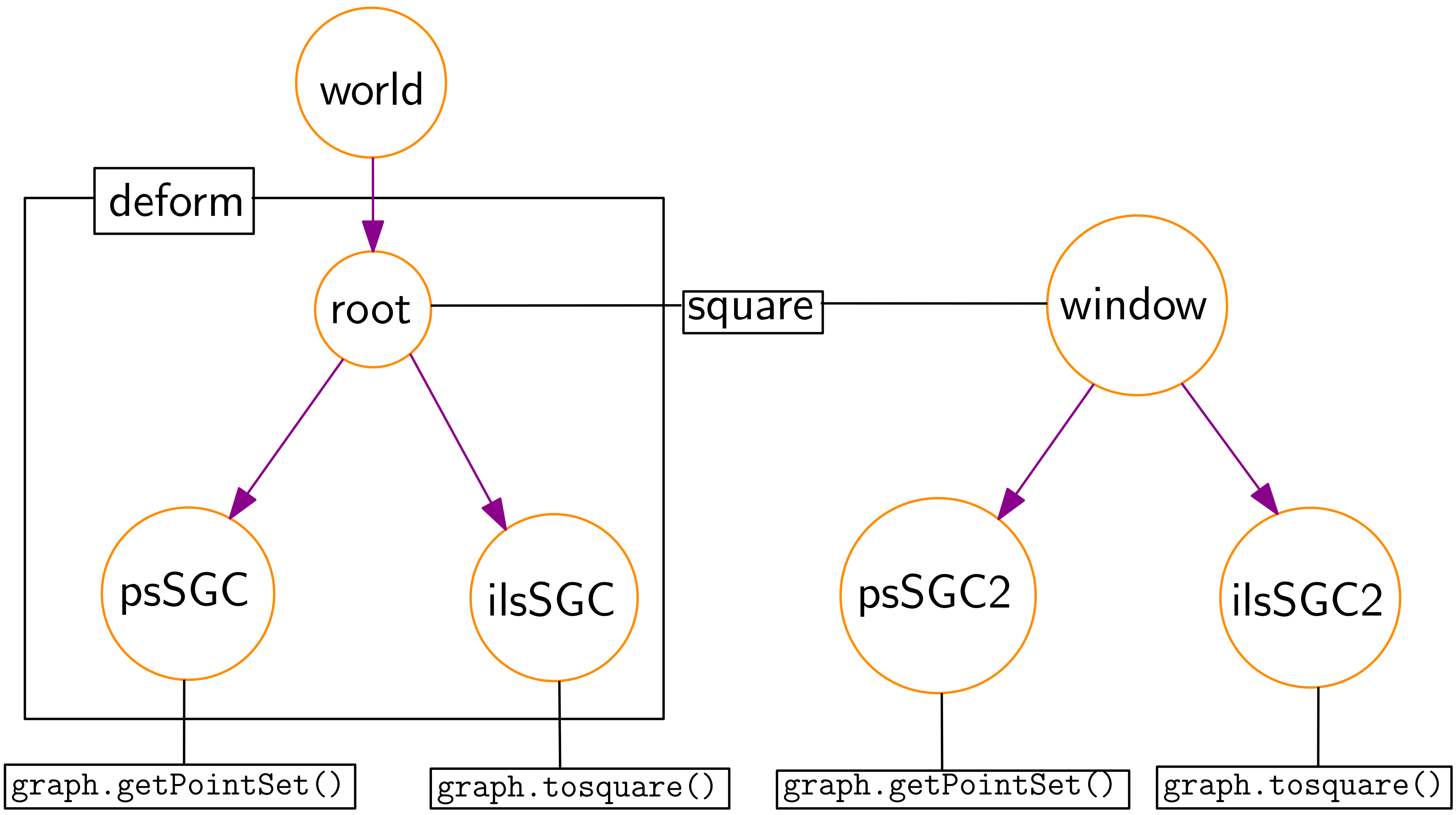
AbstractDeformation

Deform:

Goes through a SceneGraphComponent and all of its children.
If its geometry is a PointSet: map them on the torus.
If its geometry is an IndexedLineSet: refine all lines, cast to PointSet and map on torus.
Change geometries of the SGCs to the new ones.

Project

SceneGraphComponents:



Project

Tools:

VertexTool (psSGC's):

- **LeftClick** activate vertex
- **Drag LeftClik** move vertex
- **RigthClick** delete vertex
- **Shift+LeftClick** create edge between active vertex and vertex

Project

Tools:

VertexTool (psSGC's):

- **LeftClick** activate vertex
- **Drag LeftClik** move vertex
- **RigthClick** delete vertex
- **Shift+LeftClick** create edge between active vertex and vertex

EdgeTool (ilsSGC's):

- **LeftClick** activate edge
- **RightClick** delete edge

Project

Tools:

VertexTool (psSGC's):
- **LeftClick** activate vertex
- **Drag LeftClik** move vertex
- **RigthClick** delete vertex
- **Shift+LeftClick** create edge between active vertex and vertex

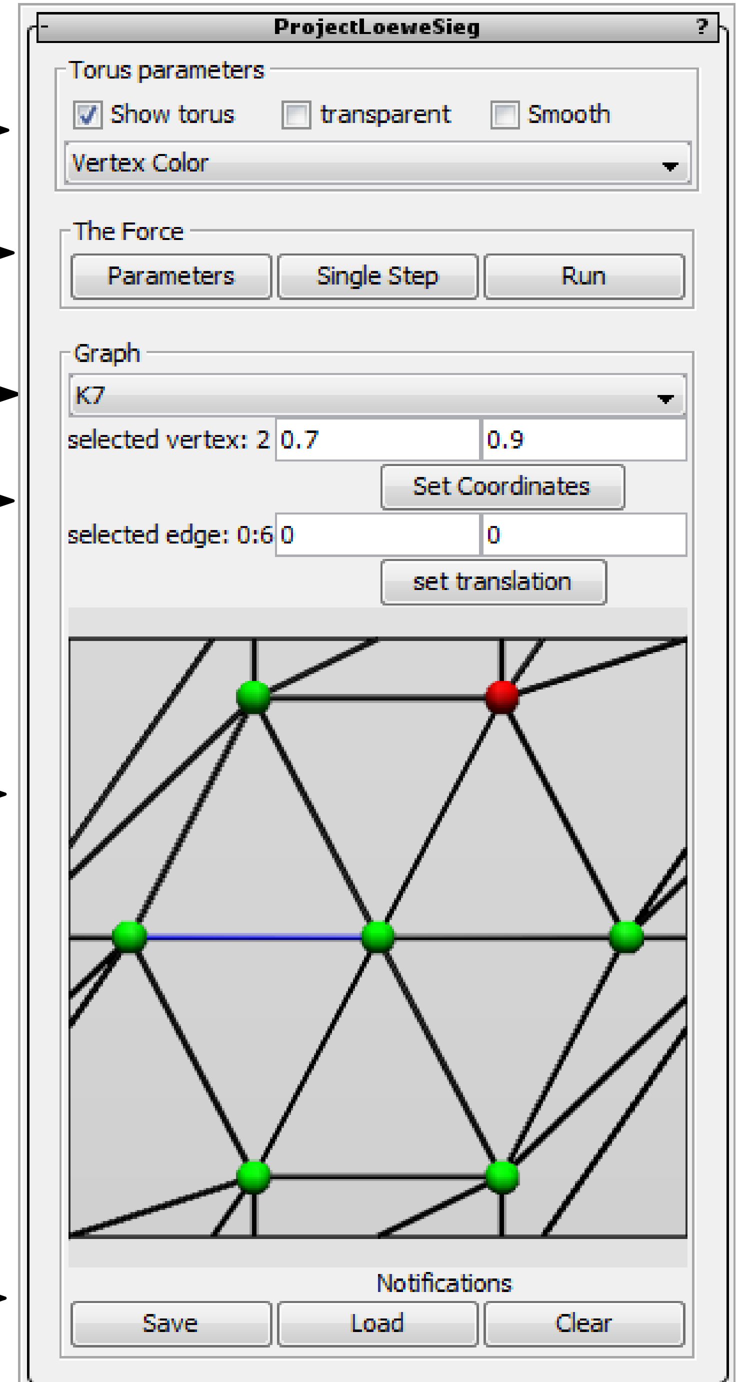
EdgeTool (ilsSGC's):
- **LeftClick** activate edge
- **RightClick** delete edge

VertexAddTool (root & window):
- **LeftClick** add vertex and activate it

Project

GUI:

- edit appearances
- run force algorithm
- choose from K_1 to K_7
- edit vertices and edges
- edit graph on flat torus
- save, load and delete graph



Presentation

Ideas:

- Implementation of other surfaces.
  $\rightarrow$ adjust the deformation and boundaries of the fundamental domain.
- Development of a *planarity game*.
  Some levels of increasing difficulty in which the user has to make a given graph planar by editing vertices and edges.
- Visualize the map given by K_7 with seven different colors.
  (Or color any other map on the torus.)

Questions? Remarks?

Questions? Remarks?

Thank you for your attention!