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 edges 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 edges the same length (*equilibrium*).

Idea:
Consider the vertices as *electrons* that push off each other and edges as springs.
The Force-Based Algorithm

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

Result:
The Force-Based Algorithm

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

Result:
The Force-Based Algorithm

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

Result:
MyGraph

Data Structure for the graph:
◦ vertices, edges, edgemove etc.
◦ adding & deleting points and edges
◦ tosquare()
◦ Force-Algorithm

Data Structure

Project

Abstract Deformation
Data Structure

MyGraph

Project

Abstract Deformation

Data Structure for the graph:
- vertices, edges, edgemove etc.
- adding & deleting points and edges
- tosquare()
- Force-Algorithm

Visualization of a graph on the torus
- Appearances
- GUI
- Tools
- Save & load graphs
Data Structure

MyGraph

Project

Abstract Deformation

Data Structure for the graph:
- vertices, edges, edgemove etc.
- adding & deleting points and edges
- tosquare()
- Force-Algorith

Visualization of a graph on the torus
- Appearances
- GUI
- Tools
- Save & load graphs

Class to deform the unit square and all its sub-geometries into the torus.
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
- `toshape()`: creates the final IndexedLineSet
- `theForce(double a, double b, double c)`: a single step of the Force-Algorithm
tosquare()
tosquare()
tosquare()

List:
\[ v_1 \, p_1 \, p_2 \]

shift point

calculate intersection point with square

translated point
tosquare()

List: $v_1, p_1, p_2, p_3$

1. Shift point $v_1$ to $p_1$.
2. Shift the translated point to $p_2$.
3. Calculate the intersection point with the square.
tosquare()
tosquare()

- Shift point
- Calculate intersection point with square
- Shift translated point
- Translated point

List:
\[
\begin{align*}
v_1 & \quad p_1 & \quad p_2 & \quad p_3 & \quad p_4 \\
p_5 & \quad p_6
\end{align*}
\]
tosquare()

List:
\[ v_1 \ p_1 \ p_2 \ p_3 \ p_4 \ \ p_5 \ p_6 \ v_2 \]

Edges:
\[ (i, i + 1) \ i \ \text{even} \]
The Force-Based Algorithm

if equilibrium = true then return
end if

for all \( v_i \in V \) do  \( \triangleright \) 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
end for

for all \( v_j \in V \) do  \( \triangleright \) Calculate Attraction
  \( \tilde{v}_j \leftarrow v_j + \text{edgemove}(v_i, v_j) \)  \( \triangleright \) 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

\( velo(i) \leftarrow k_3 \cdot (velo(i) + net(i)) \)
The Force-Based Algorithm

for all $v_i \in V$ do
  if $\|velo(i)\| > \varepsilon$ then break
end if
  if at last vertex then
    equilibrium=true
    return
  end if
end for

for all $v_i \in V$ do
  $v_i \leftarrow v_i + velo(i)$  
  ▷ 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:

- world
- root
- world
- psSGC
- ilsSGC
- window
- psSGC2
- ilsSGC2

- deform
- square
- graph.getPointSet()
- graph.tosquare()
### Vertex Tool (psSGC’s):

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

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

**EdgeTool** (ilsSGC’s):
- **LeftClick** activate edge
- **RightClick** delete edge
### Project

#### Tools:

**Vertex Tool (psSGC’s):**
- **LeftClick** activate vertex
- **Drag LeftClick** move vertex
- **RightClick** delete vertex
- **Shift+LeftClick** create edge between active vertex and vertex

**Edge Tool (ilsSGC’s):**
- **LeftClick** activate edge
- **RightClick** delete edge

**Vertex Add Tool (root & window):**
- **LeftClick** add vertex and activate it
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
Outlook

Idea:

◦ Implementation of other surfaces.
  → 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!