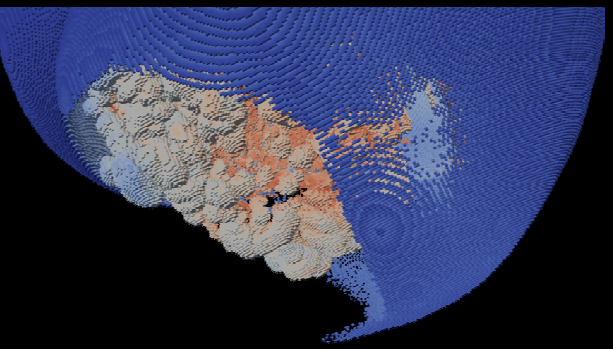
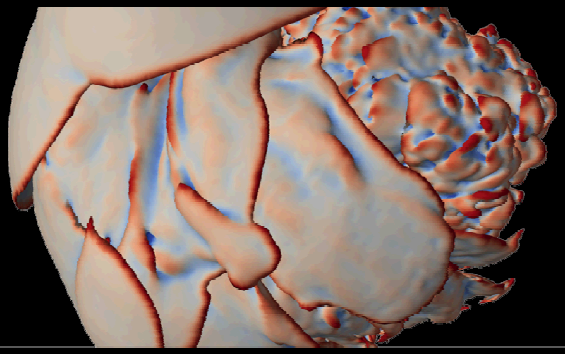
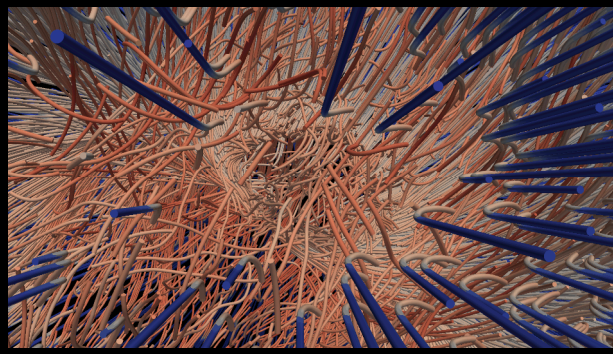
Exceptional service in the national interest

SAND2012-8219P

Sandia National Laboratories







Dax: A Pervasively Parallel Framework for Data Analysis and Visualization

Kenneth Moreland, Sandia National Laboratories

October 3, 2012

U.S. DEPARTMENT OF ENERGY

National Nuclear Security Administration

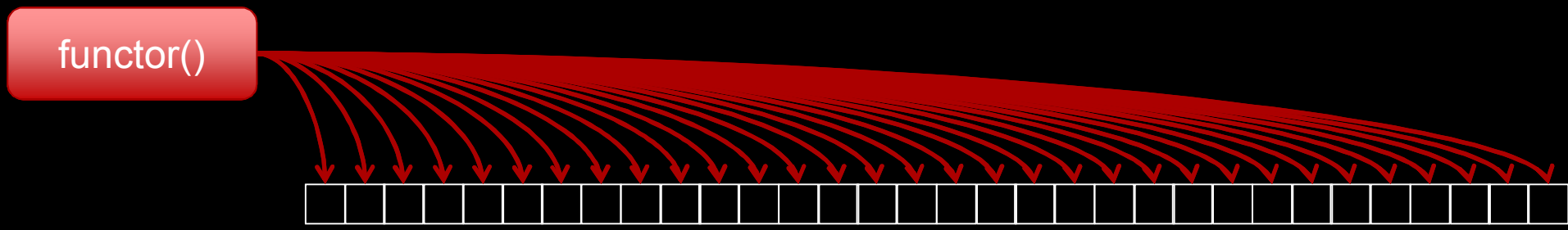
Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

Project Goals

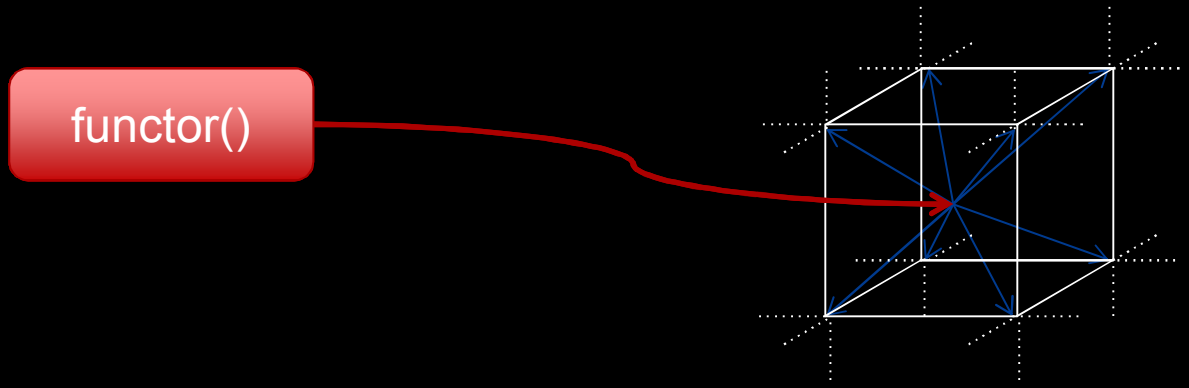
- Develop readiness for scientific data analysis and visualization at extreme scale.
  - In particular, address the challenges of emerging multi- and many-core architectures.
- Create a toolkit that well suited to design of visualization operations with a great number of shared memory threads.
- Develop a framework that adapts to emerging processor and compiler technologies.
- Design multi-purpose algorithms that can be applied to a variety of visualization operations.

Basic Approach

- Functor mapping [Baker, et al. 2010]

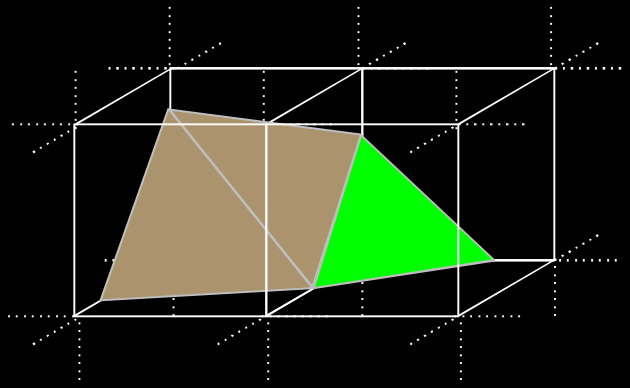


Applied to Topologies

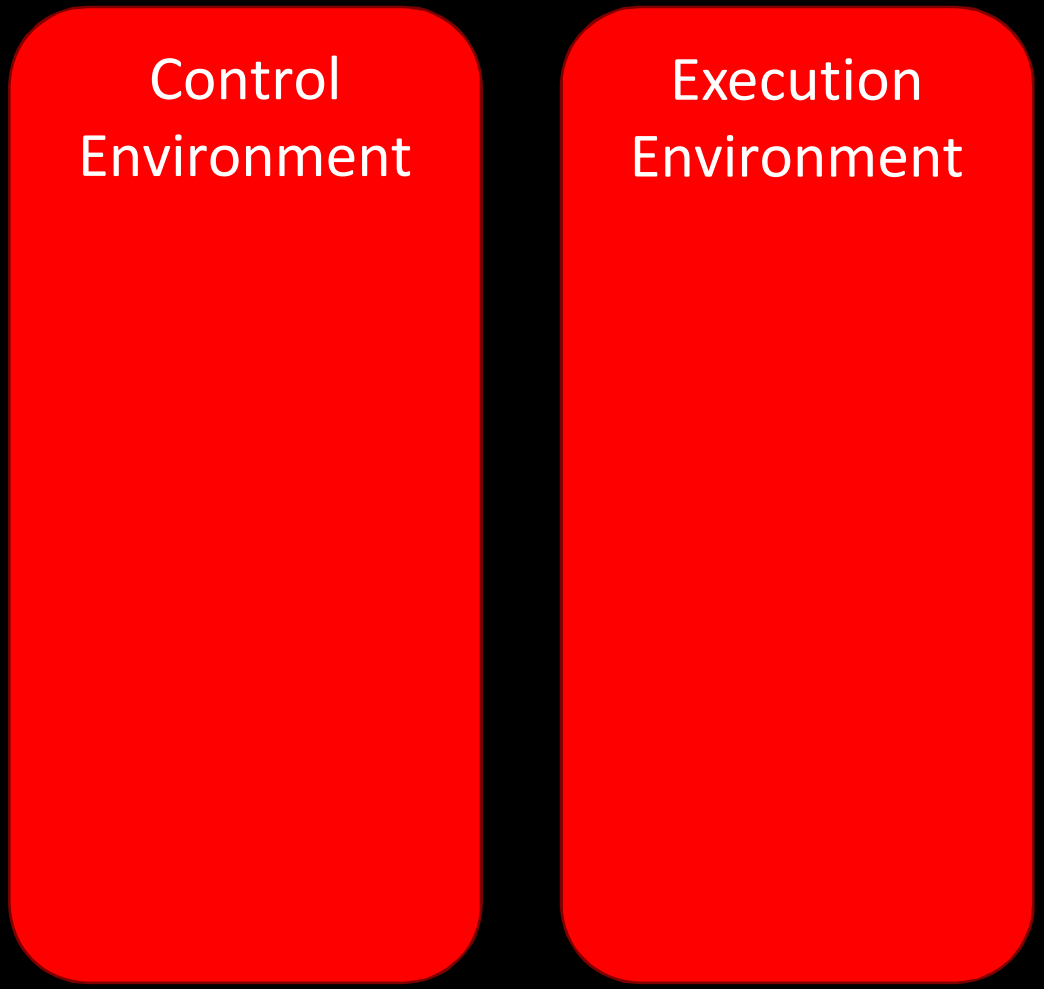


Applied to Topologies

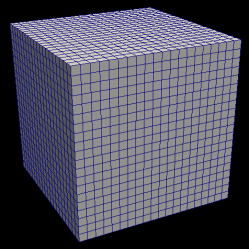
functor()



Dax Framework



Dax Framework



Control Environment

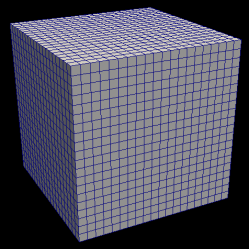
Grid Topology
Array Handle
Invoke

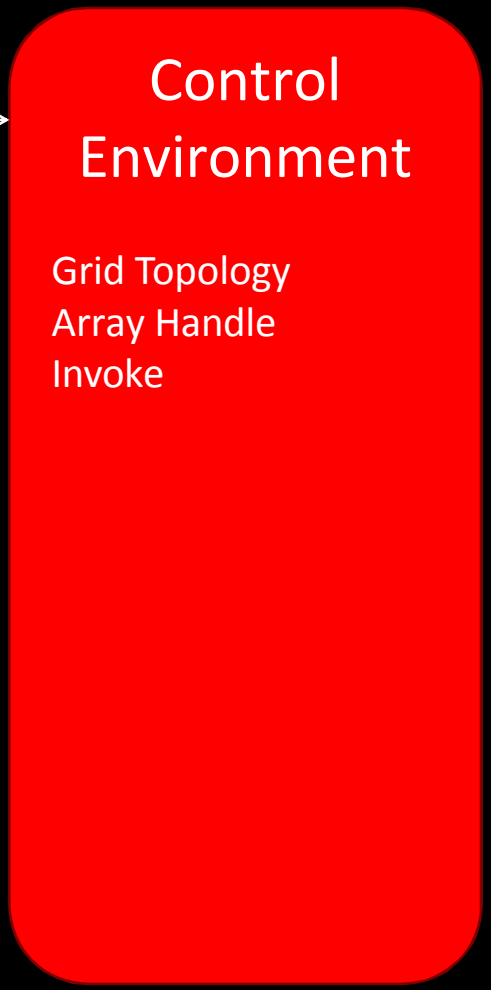
dax::cont

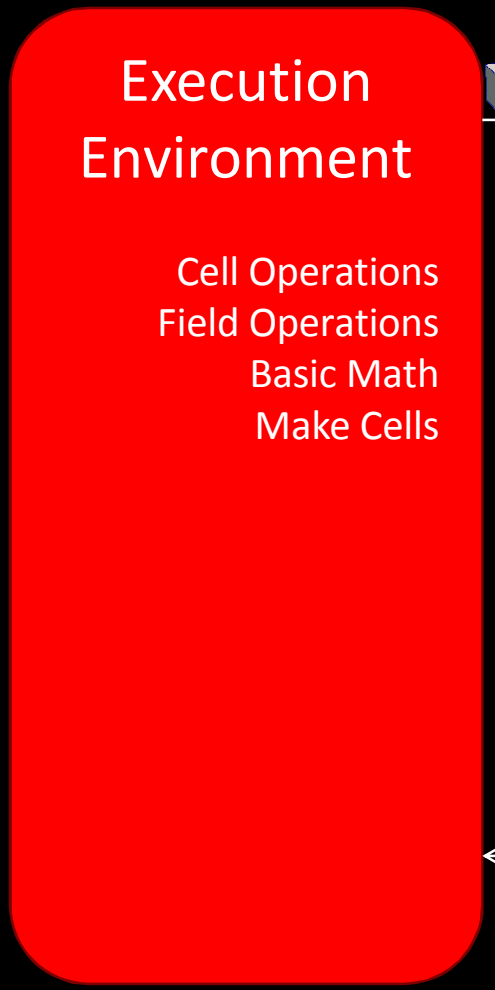
Execution Environment

dax::exec

Dax Framework



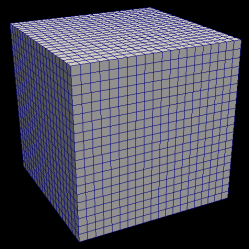




dax::cont

dax::exec

Dax Framework



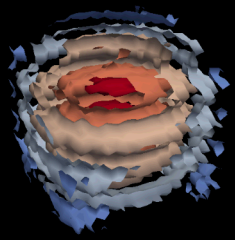
Control Environment

Grid Topology
Array Handle
Invoke

Execution Environment

Cell Operations
Field Operations
Basic Math
Make Cells

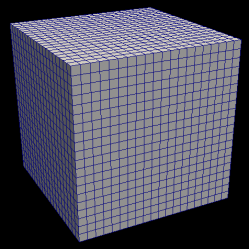
Worklet



dax::cont

dax::exec

Dax Framework



Control Environment

Grid Topology
Array Handle
Invoke

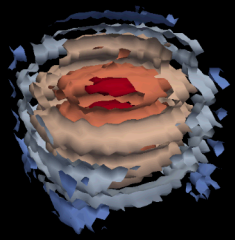
Device Adapter

Allocate
Transfer
Schedule
Sort
…

Execution Environment

Cell Operations
Field Operations
Basic Math
Make Cells

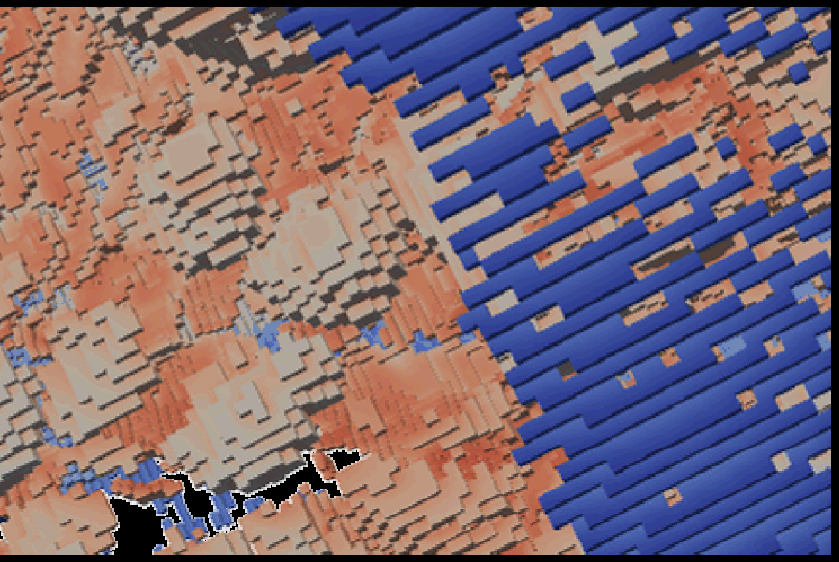
Worklet



dax::cont

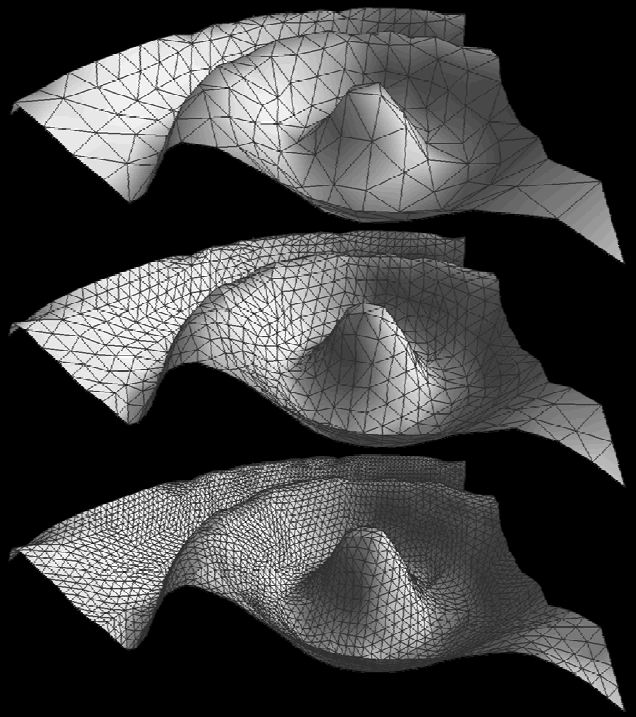
dax::exec

Threshold (with point removal)



Contour (with normal estimation and surface improvements)

Subdivision (with quadratic smoothing)



Contour (with normal, surface improvements, quadratic smoothing, and curvature estimation)

```
template<typename ValueType>
class ThresholdClassify : public dax::exec::WorkletMapCell
{
public:
  typedef void ControlSignature(Topology, Field(Point), Field(Out));
  typedef _3 ExecutionSignature(_1,_2);

  template<class InputCellType>
  DAX_EXEC_EXPORT dax::Id operator()(
      InputCellType,
      const dax::Tuple<ValueType,InputCellType::NUM_POINTS> &values) const
  {
    ThresholdFunction<ValueType> threshold(this->ThresholdMin, this->ThresholdMax);
    dax::exec::VectorForEach(values, threshold);
    return threshold.valid;
  }
};


class ThresholdTopology : public dax::exec::WorkletGenerateTopology
{
public:
  typedef void ControlSignature(Topology, Topology(Out));
  typedef void ExecutionSignature(Topology::PointIds(_1), Topology::PointIds(_2));

  template<int NumInputPoints, int NumOutputPoints>
  DAX_EXEC_EXPORT void operator()(dax::Tuple<dax::Id,NumInputPoints> const& in,
                                  dax::Tuple<dax::Id,NumOutputPoints> &out) const
  {
    out = in;
  }
};
```

```
template<typename ValueType>
class ThresholdClassify : public dax::exec::WorkletMapCell
{
public:
  typedef void ControlSignature(Topology, Field(Point), Field(Out));
  typedef _3 ExecutionSignature(_1,_2);

  template<class InputCellType>
  DAX_EXEC_EXPORT dax::Id operator()(
      InputCellType,
      const dax::Tuple<ValueType,InputCellType::NUM_POINTS> &values) const
  {
    ThresholdFunction<ValueType> threshold(this->ThresholdMin, this->ThresholdMax);
    dax::exec::VectorForEach(values, threshold);
    return threshold.valid;
  }
};
```

Report cells to be generated.

```
class ThresholdTopology : public dax::exec::WorkletGenerateTopology
{
public:
  typedef void ControlSignature(Topology, Topology(Out));
  typedef void ExecutionSignature(Topology::PointIds(_1), Topology::PointIds(_2));

  template<int NumInputPoints, int NumOutputPoints>
  DAX_EXEC_EXPORT void operator()(dax::Tuple<dax::Id,NumInputPoints> const& in,
                                  dax::Tuple<dax::Id,NumOutputPoints> &out) const
  {
    out = in;
  }
};
```

Define cells.

```
class HalfSpaceClassify : public dax::exec::WorkletMapCell
{
public:
  typedef void ControlSignature(Topology, Field(Point), Field(Out));
  typedef _3 ExecutionSignature(_1,_2);

  template<class InputCellType>
  DAX_EXEC_EXPORT dax::Id operator()(
      InputCellType,
      const dax::Tuple<dax::Vector3,InputCellType::NUM_POINTS> &values) const
  {
    for (int index = 0; index < InputCellType::NUM_POINTS; index++)
      {
      if (dax::dot(values[index],this->normal) + offset > 0) { return 1; }
      }
    return 0;
  }
};
```

Report cells to be generated.

```
class ThresholdTopology : public dax::exec::WorkletGenerateTopology
{
public:
  typedef void ControlSignature(Topology, Topology(Out));
  typedef void ExecutionSignature(Topology::PointIds(_1), Topology::PointIds(_2));

  template<int NumInputPoints, int NumOutputPoints>
  DAX_EXEC_EXPORT void operator()(dax::Tuple<dax::Id,NumInputPoints> const& in,
                                  dax::Tuple<dax::Id,NumOutputPoints> &out) const
  {
    out = in;
  }
};
```

Define cells.

```
class SphereClassify : public dax::exec::WorkletMapCell
{
public:
  typedef void ControlSignature(Topology, Field(Point), Field(Out));
  typedef _3 ExecutionSignature(_1,_2);

  template<class InputCellType>
  DAX_EXEC_EXPORT dax::Id operator()(
      InputCellType,
      const dax::Tuple<dax::Vector3,InputCellType::NUM_POINTS> &values) const
  {
    for (int index = 0; index < InputCellType::NUM_POINTS; index++)
      {
      if (Magnitude(values[index]-this->Center) < this->Radius) { return 1; }
      }
    return 0;
  }
};
```

Report cells to be generated.

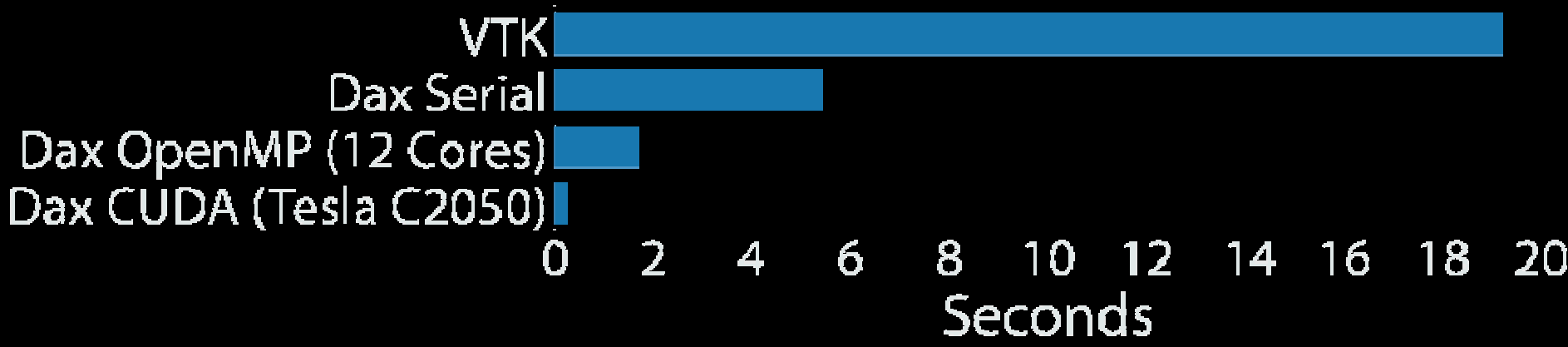
```
class ThresholdTopology : public dax::exec::WorkletGenerateTopology
{
public:
  typedef void ControlSignature(Topology, Topology(Out));
  typedef void ExecutionSignature(Topology::PointIds(_1), Topology::PointIds(_2));

  template<int NumInputPoints, int NumOutputPoints>
  DAX_EXEC_EXPORT void operator()(dax::Tuple<dax::Id,NumInputPoints> const& in,
                                  dax::Tuple<dax::Id,NumOutputPoints> &out) const
  {
    out = in;
  }
};
```

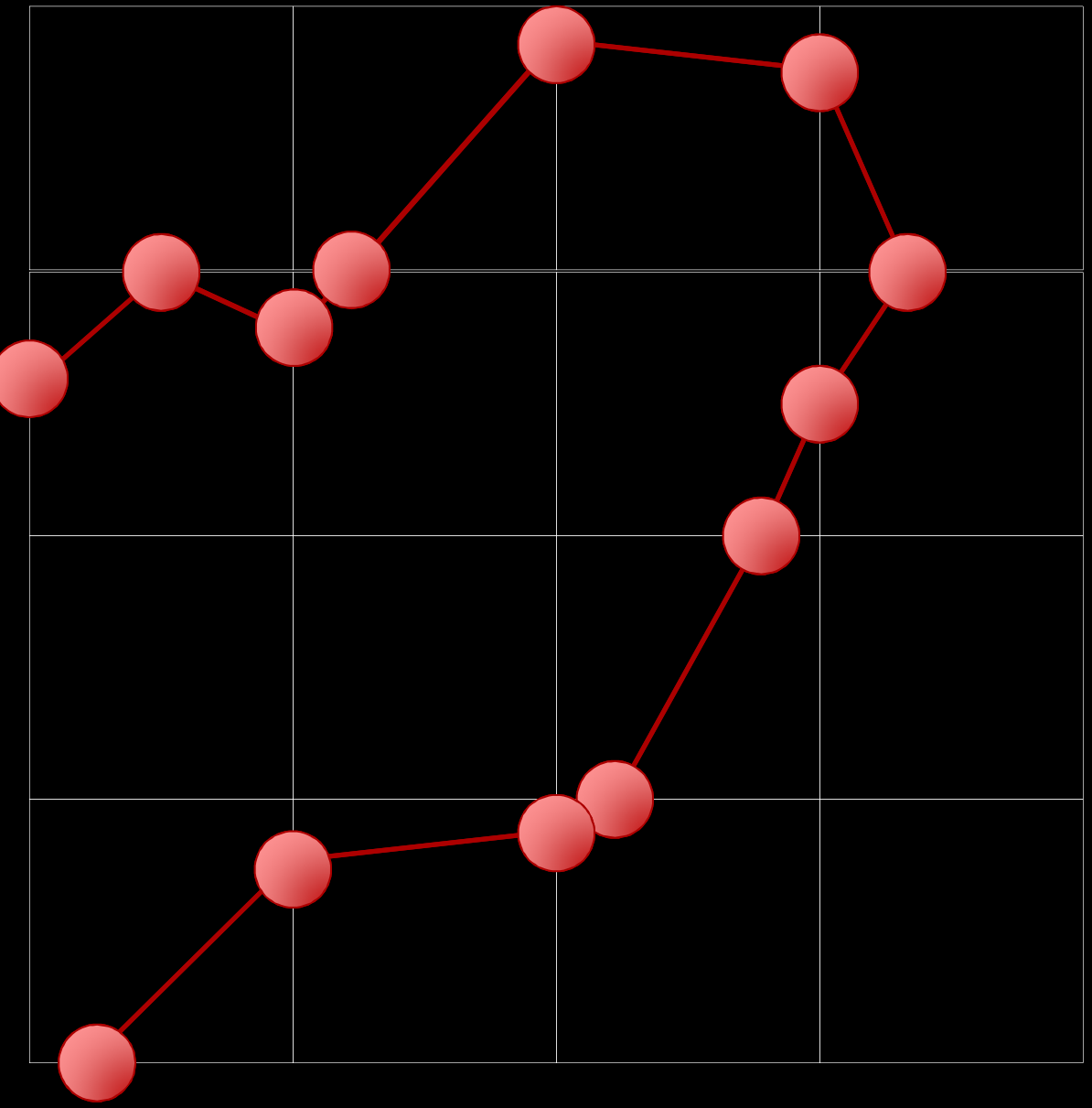
Define cells.

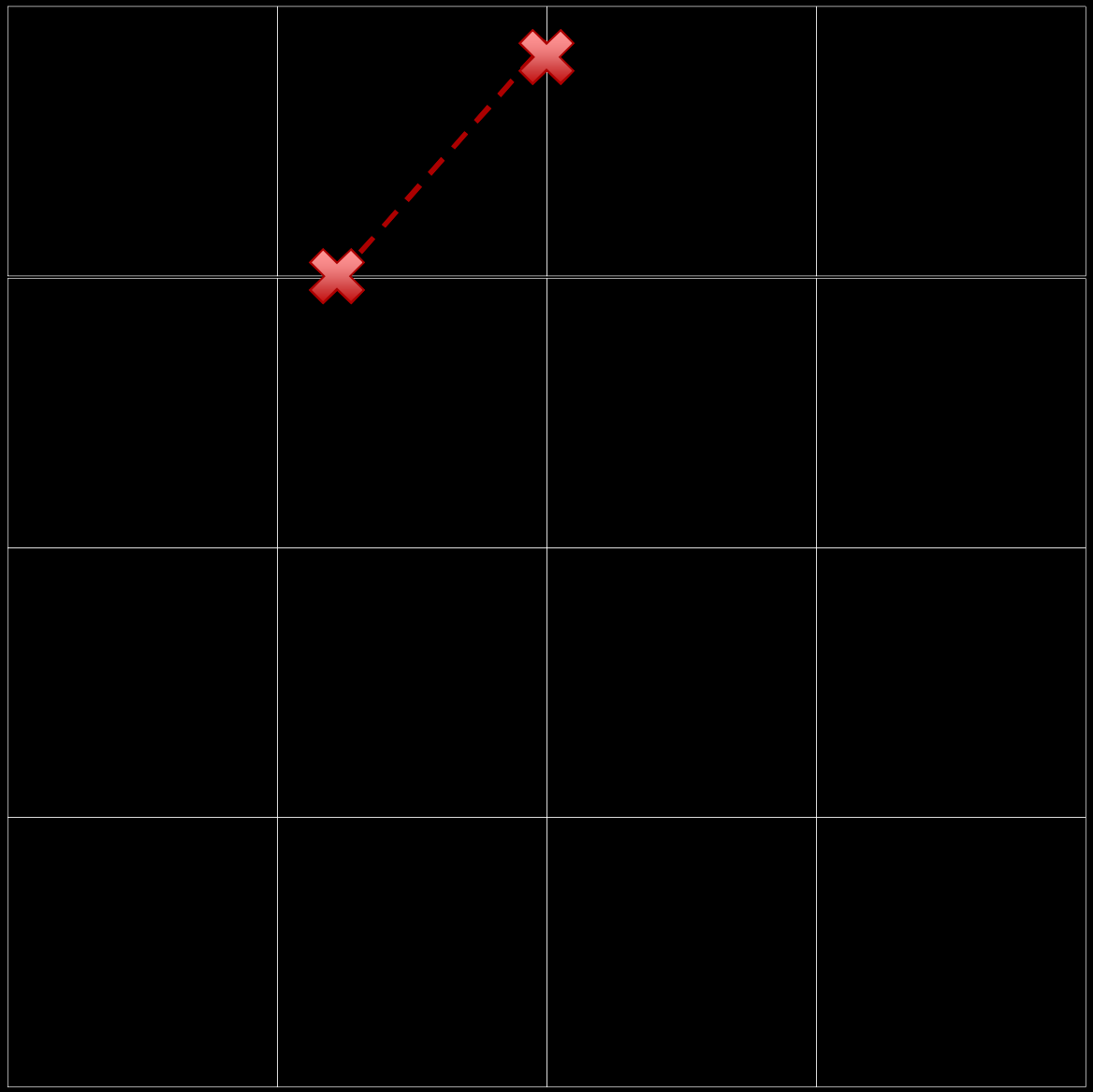
```
class TetrahedraTopology : public dax::exec::WorkletGenerateTopology
{
public:
  typedef void ControlSignature(Topology, Topology(Out));
  typedef void ExecutionSignature(
      Topology::PointIds(_1), Topology::SubIndex(_1), Topology::PointIds(_2));

  template<int NumInputPoints, int NumOutputPoints>
  DAX_EXEC_EXPORT void operator()(dax::Tuple<dax::Id,NumInputPoints> const& in,
                                  int subIndex,
                                  dax::Tuple<dax::Id,NumOutputPoints> &out) const
  {
    for (int index = 0; index < NumOutputPoints; index++)
      {
      out[index] = in[TetraVerts[index]];
      }
  }
};
```

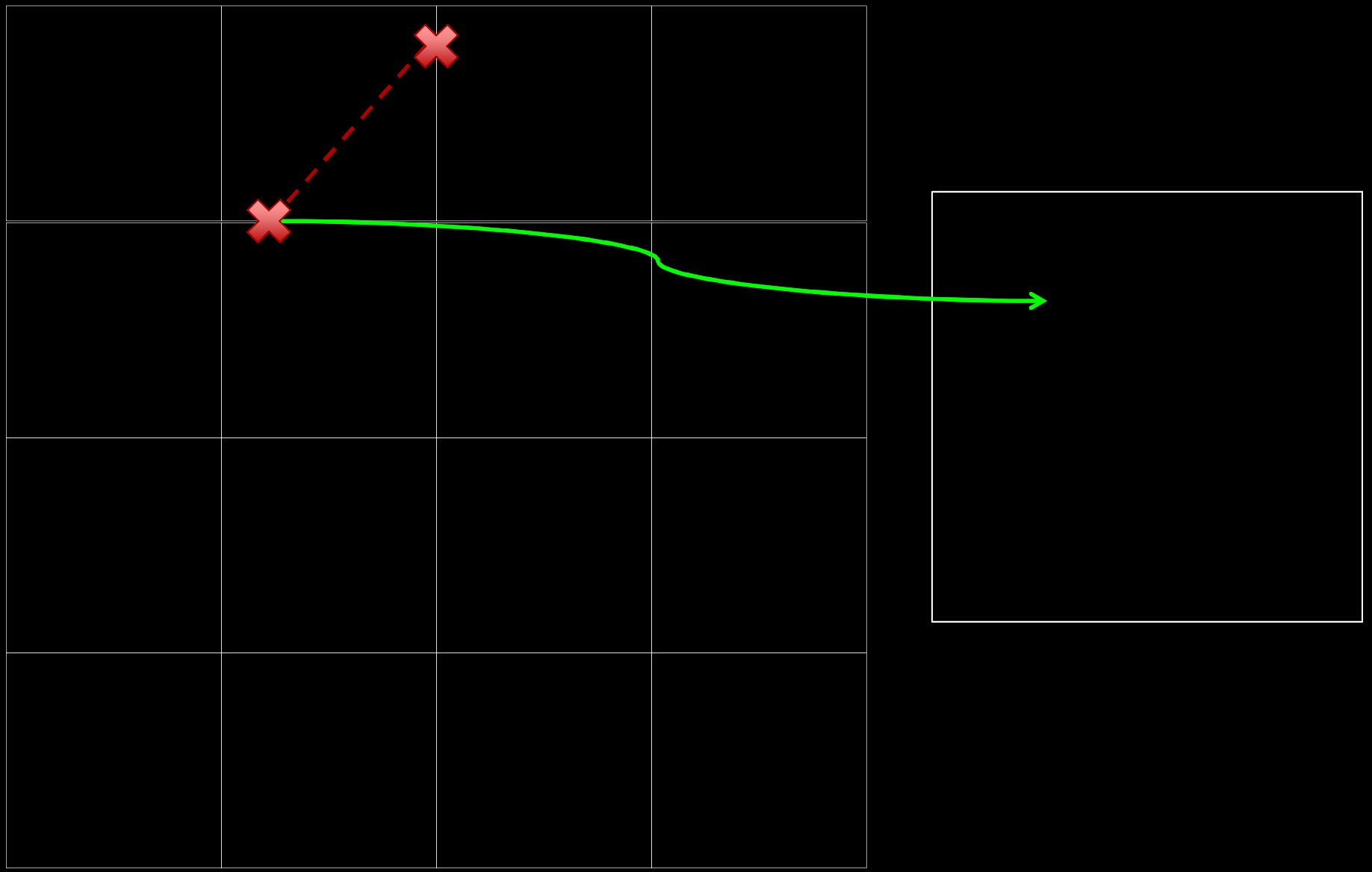
Define cells.

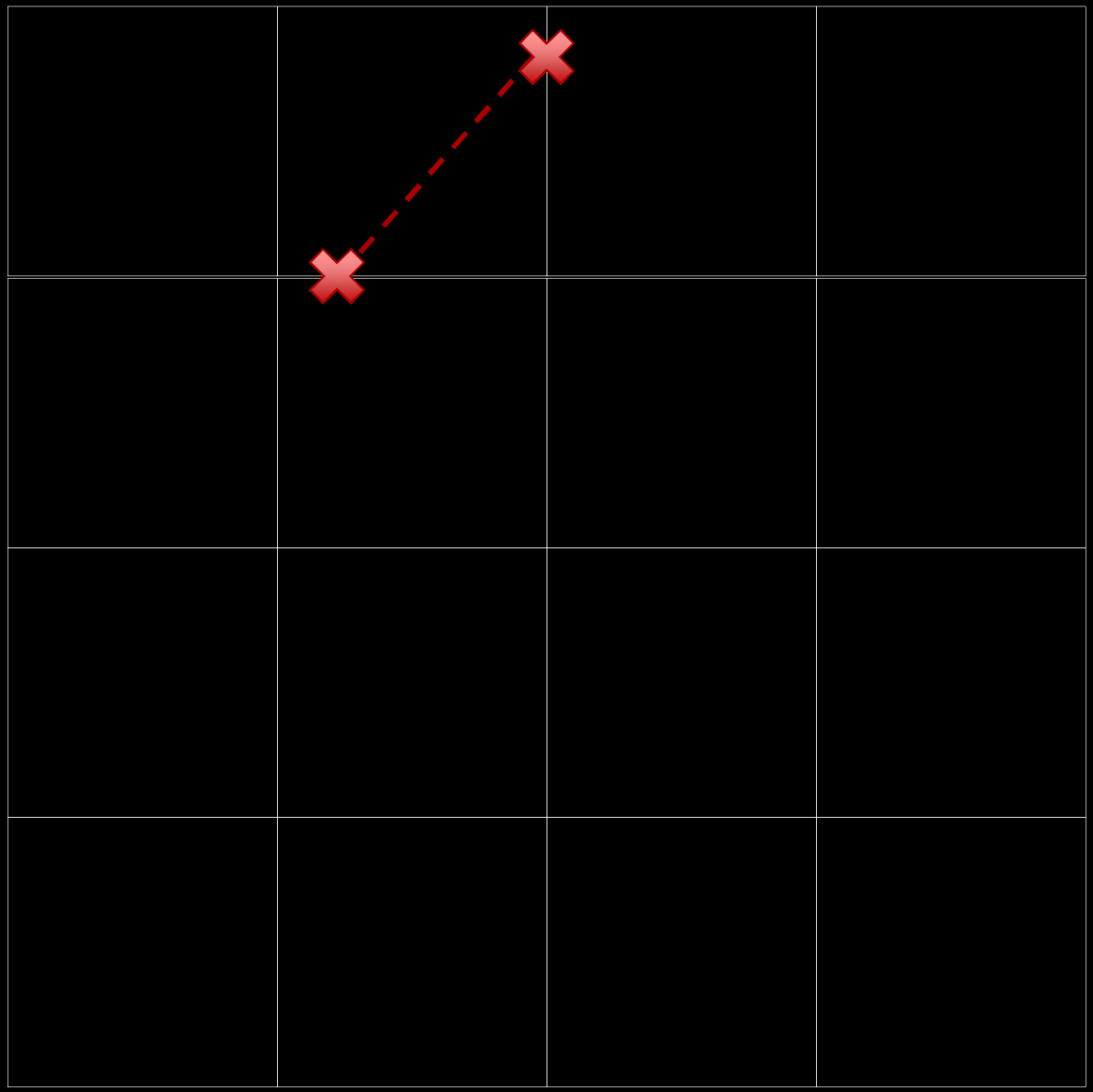
Threshold Operation with Unused Point Removal

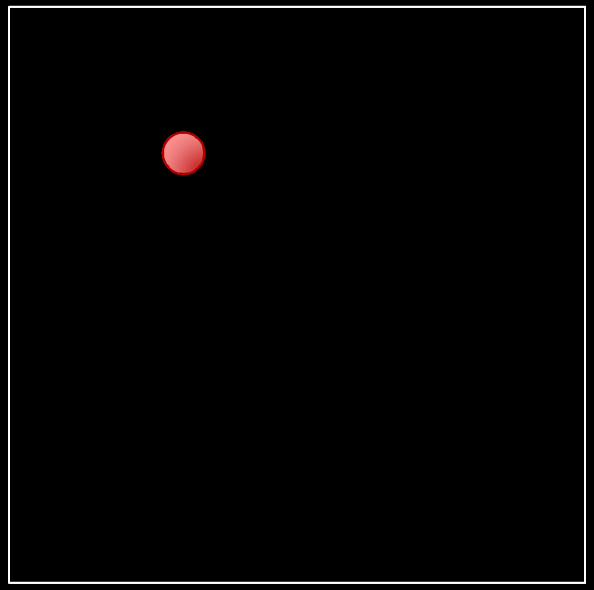


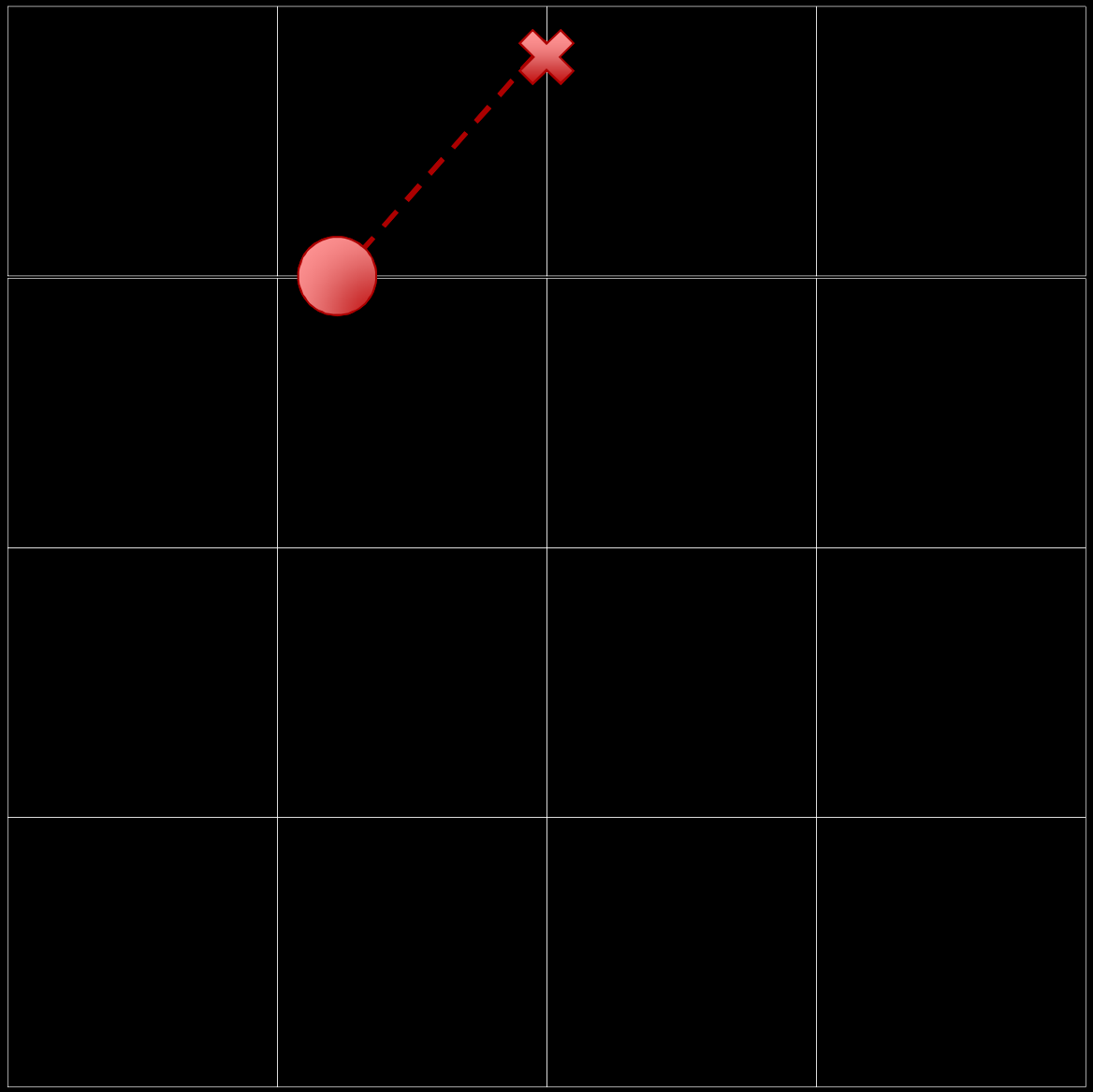


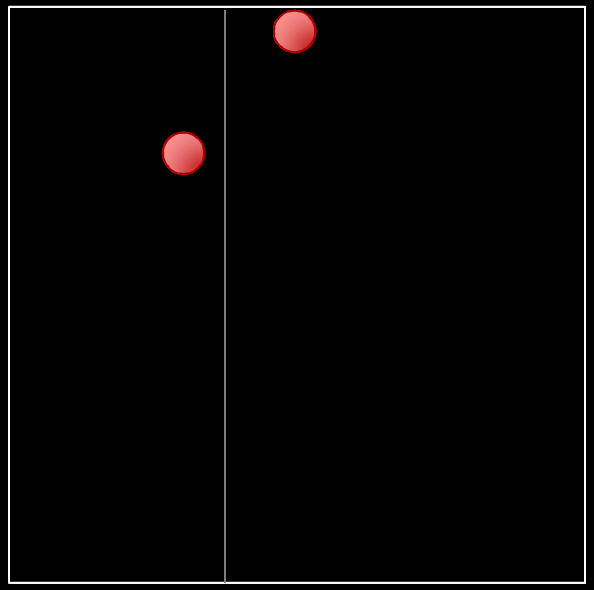


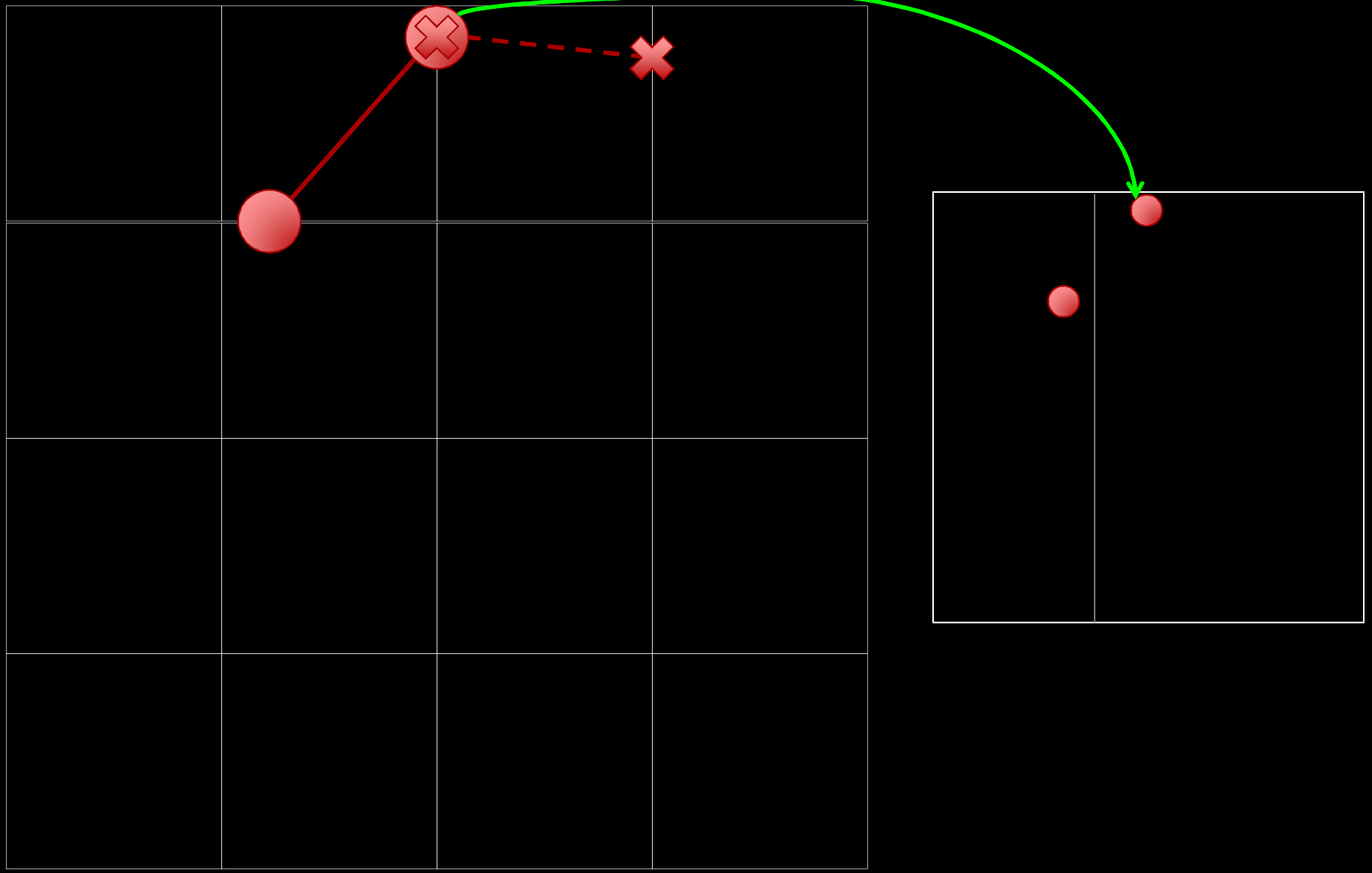


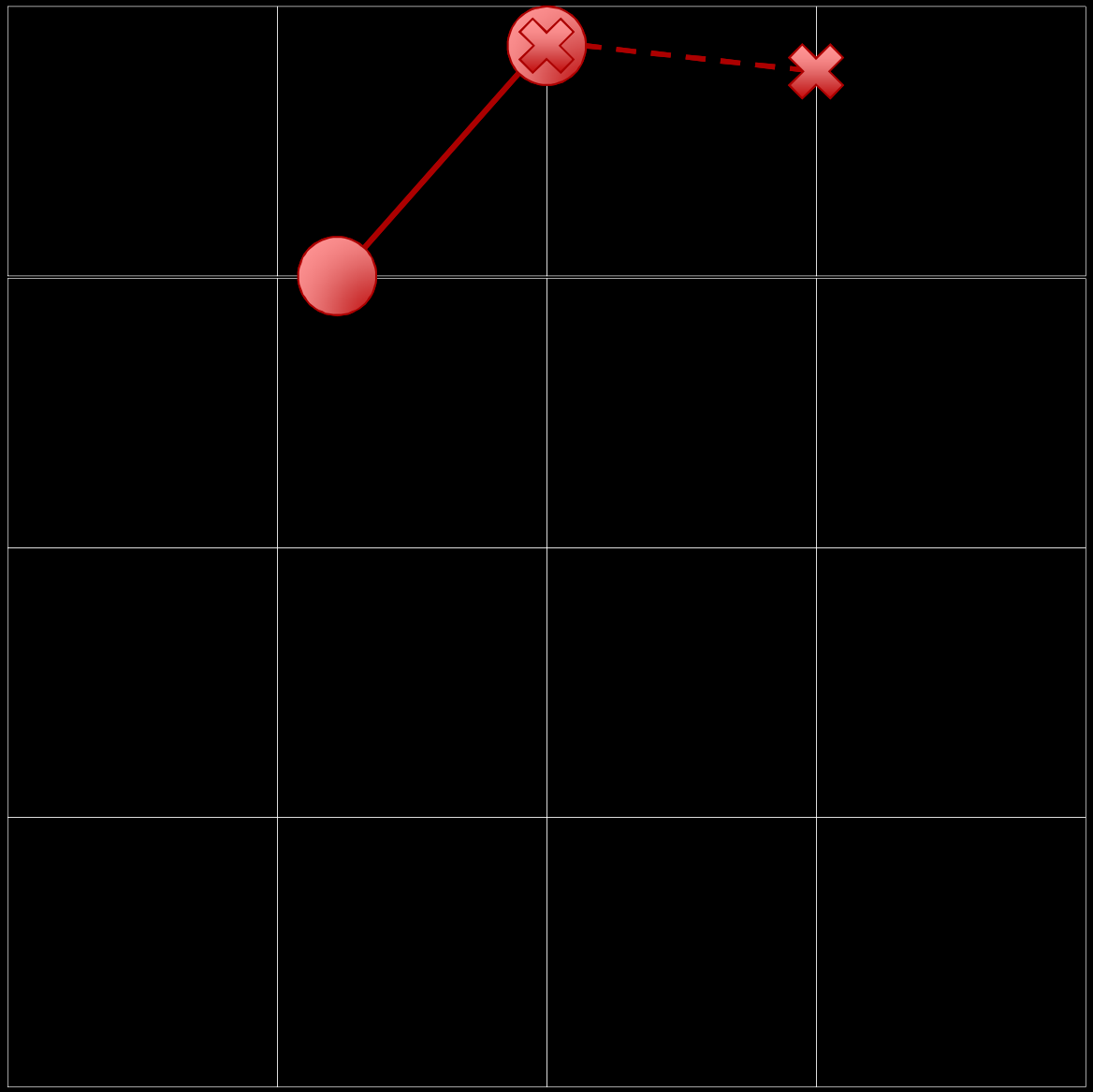


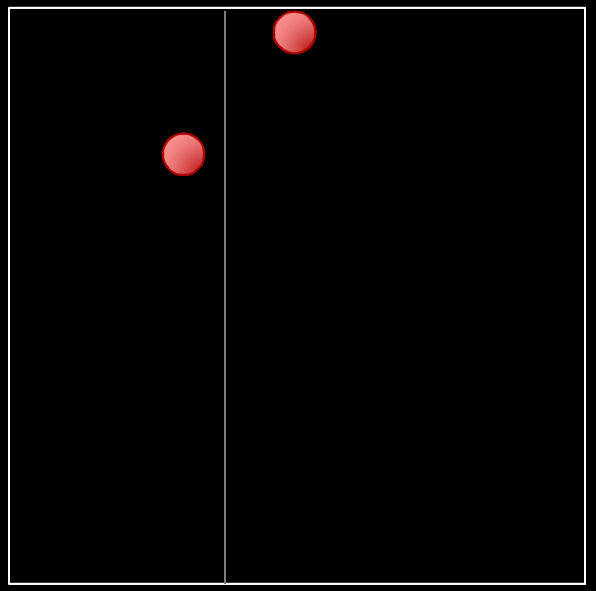


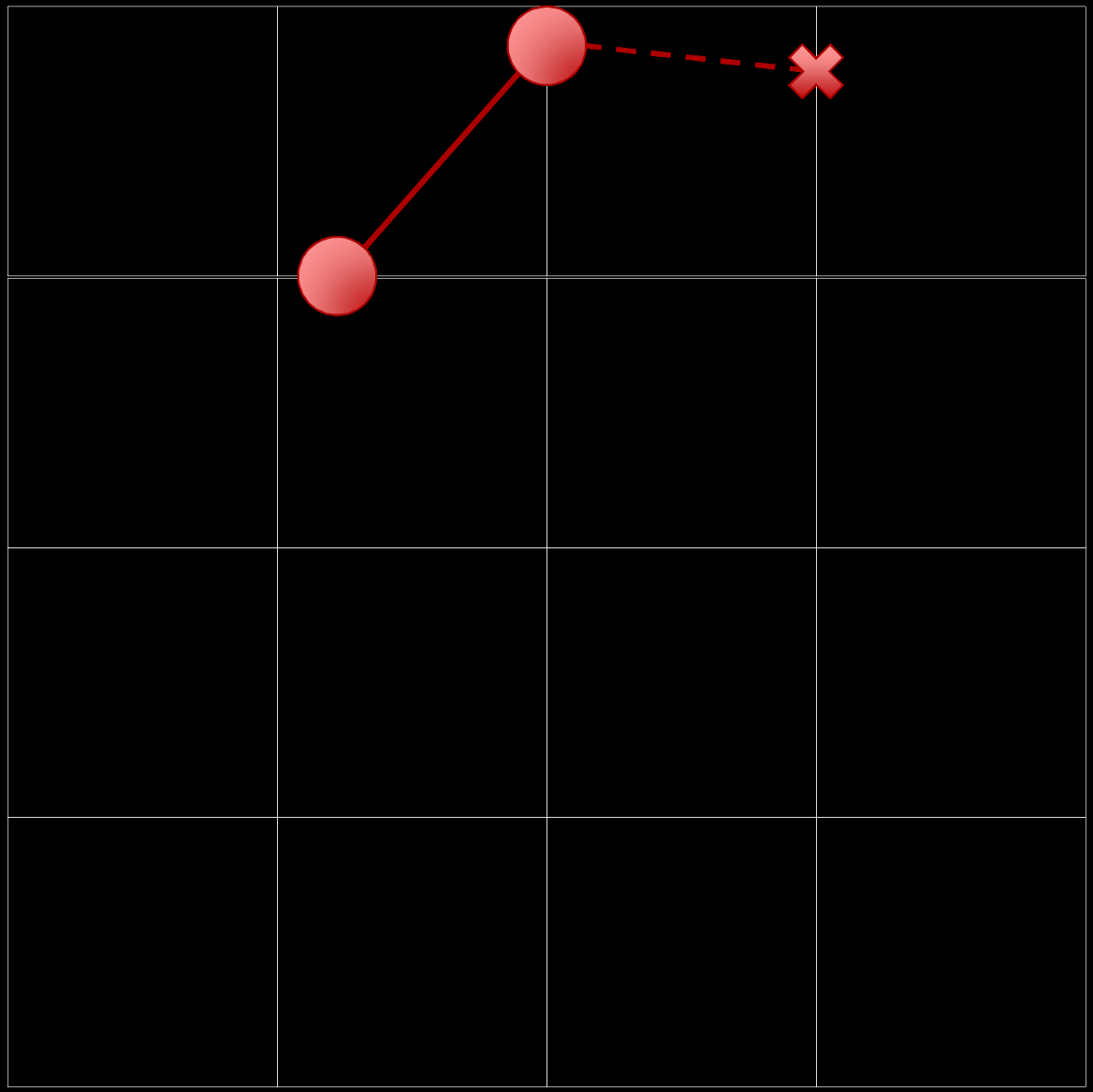


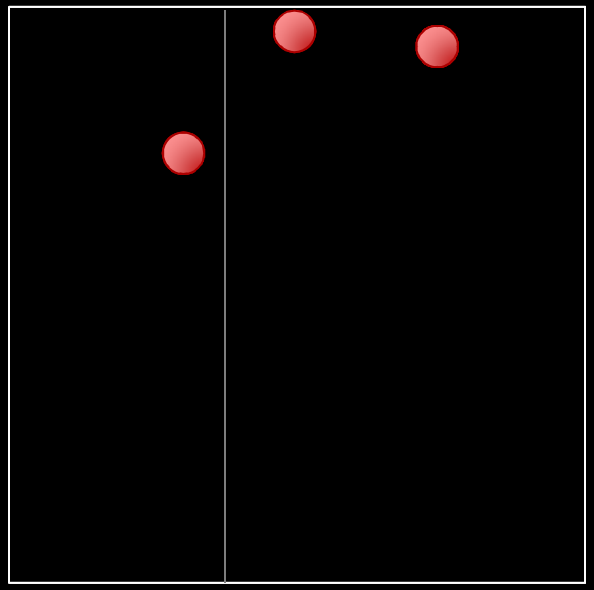


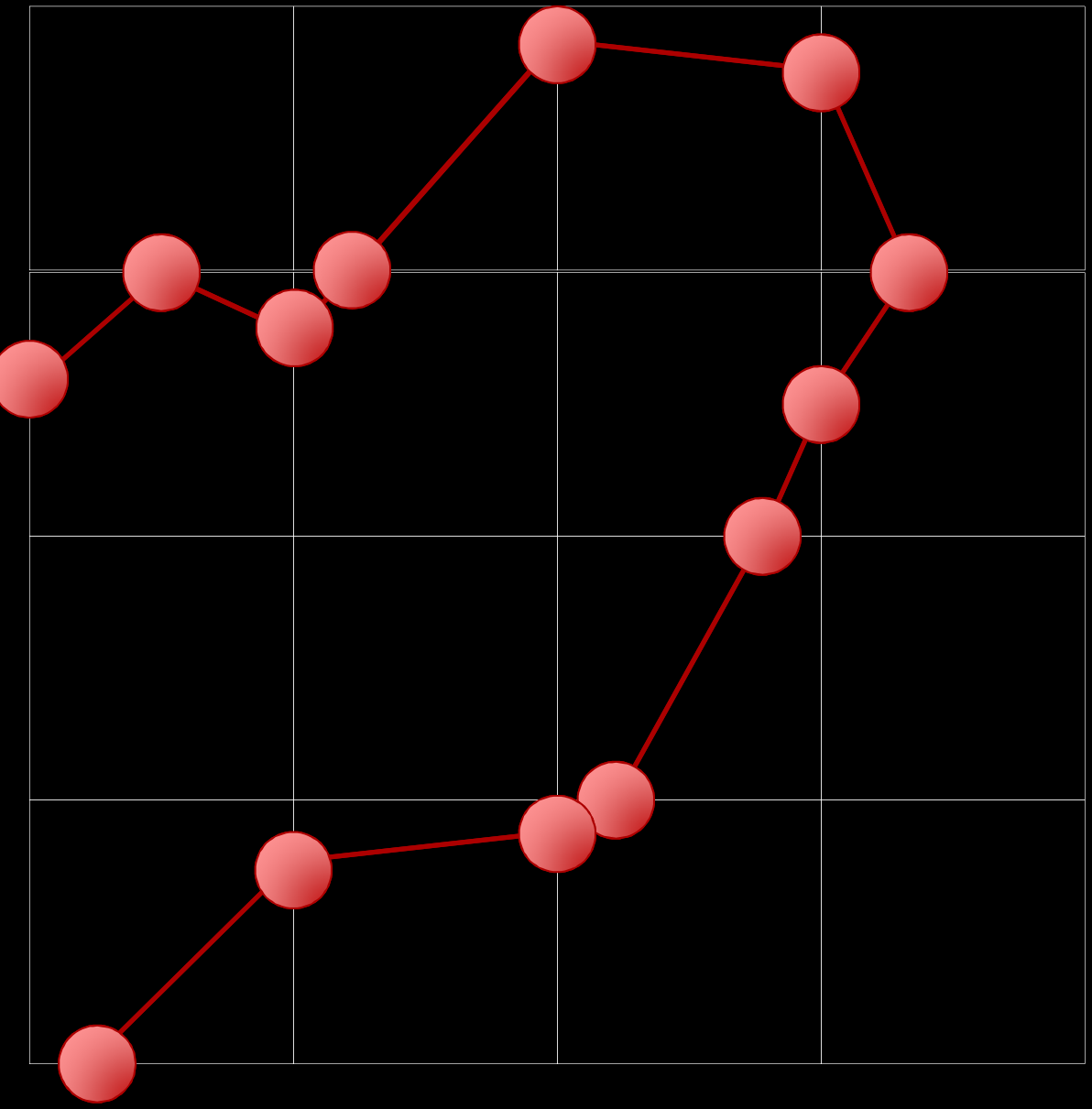


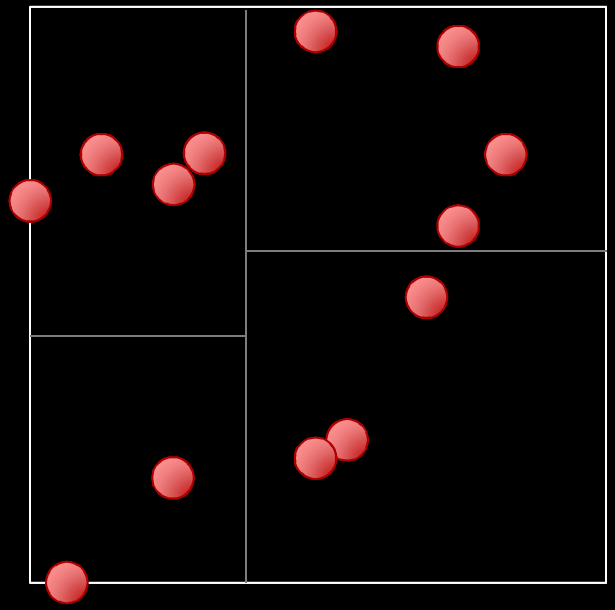


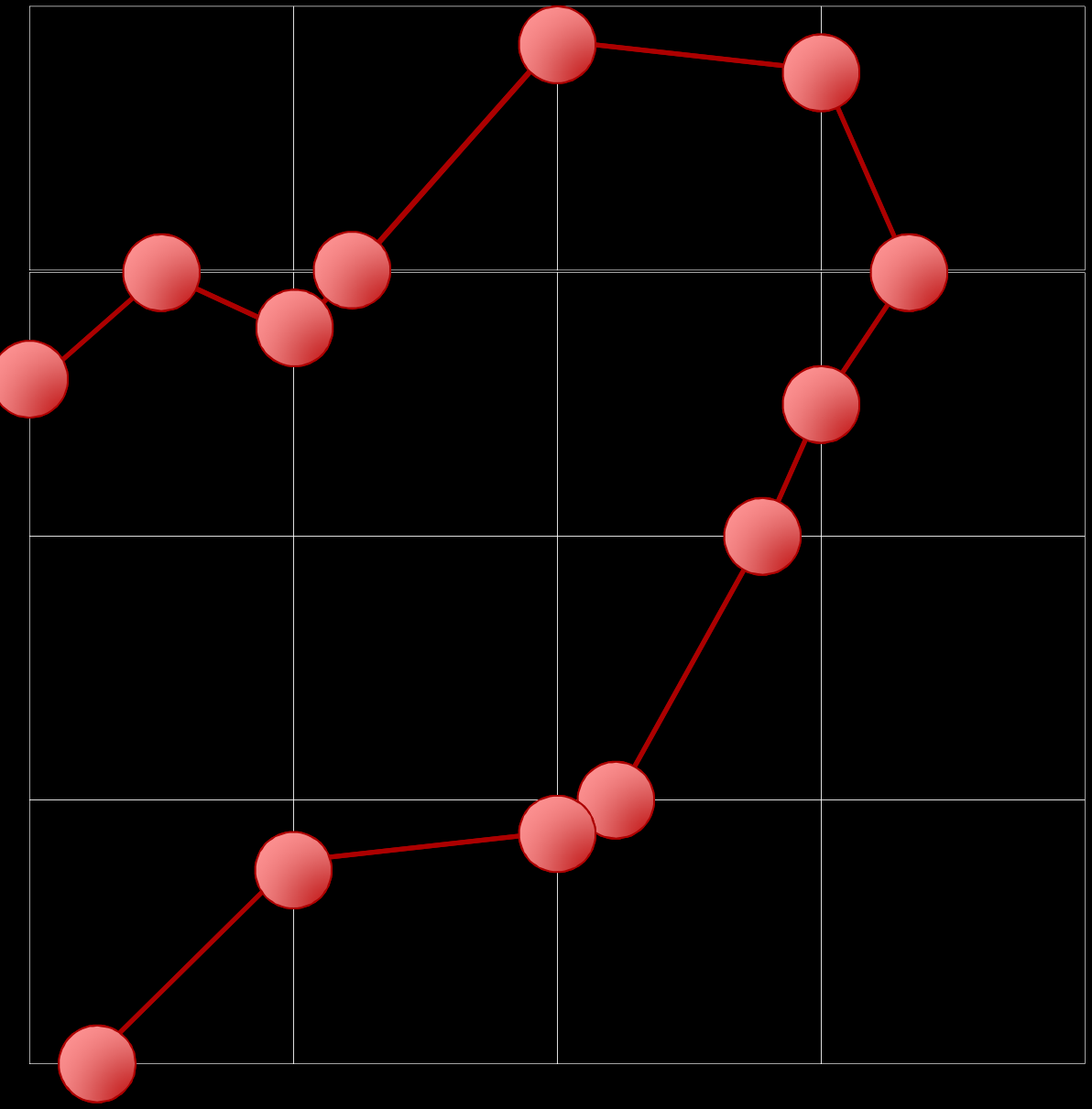


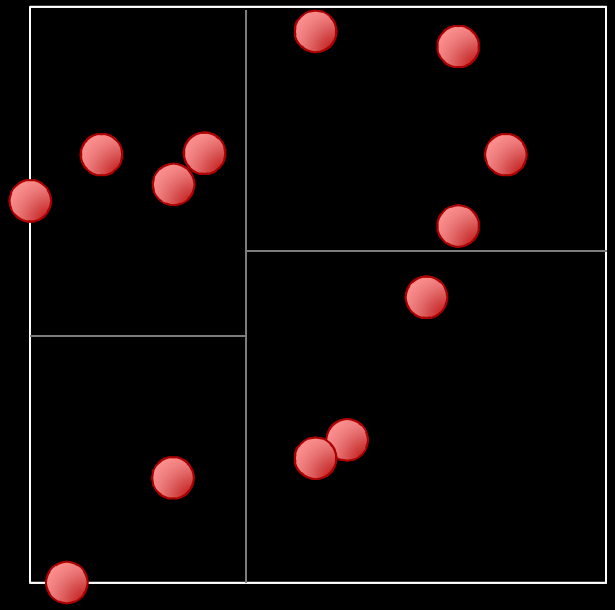


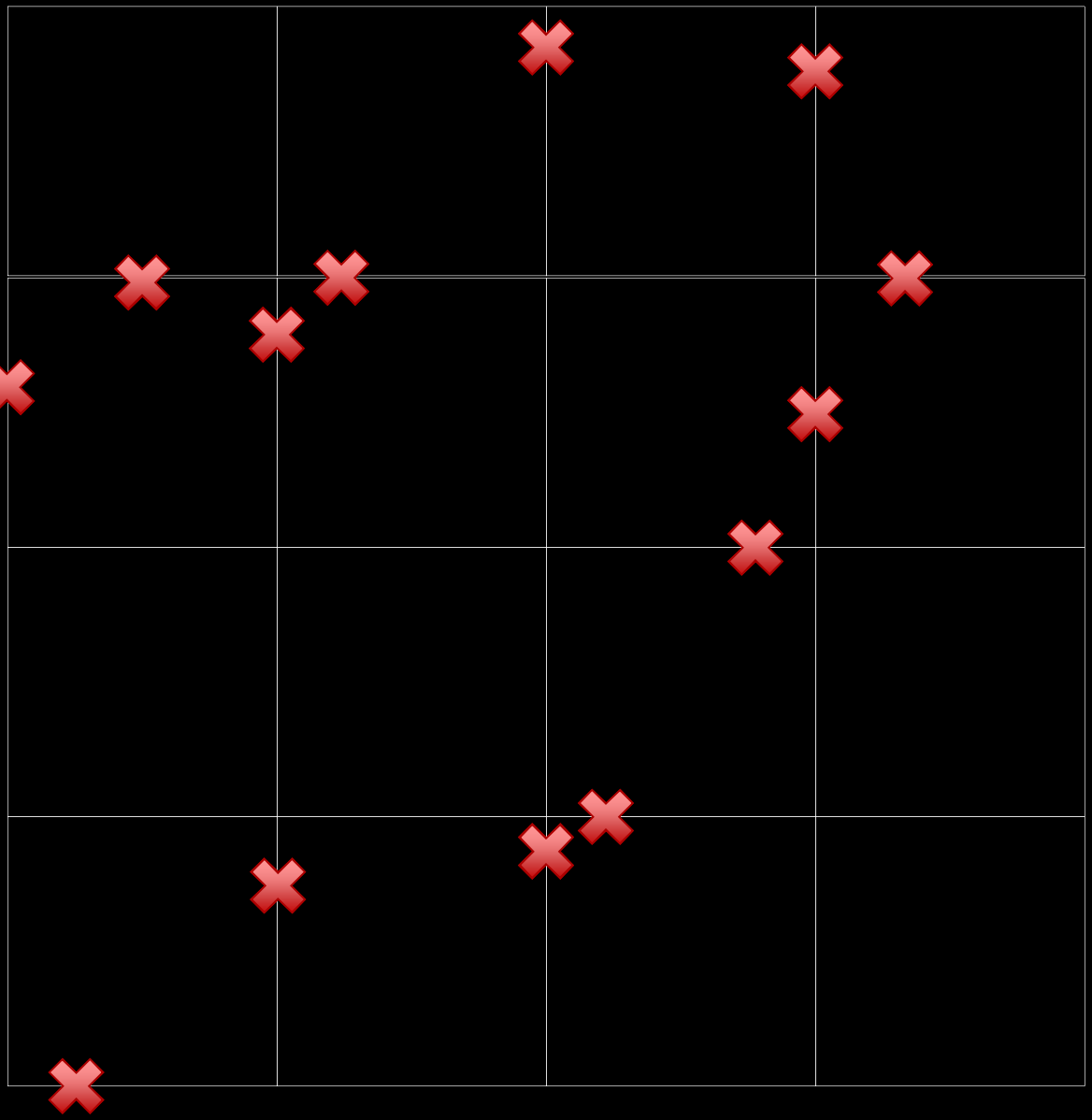


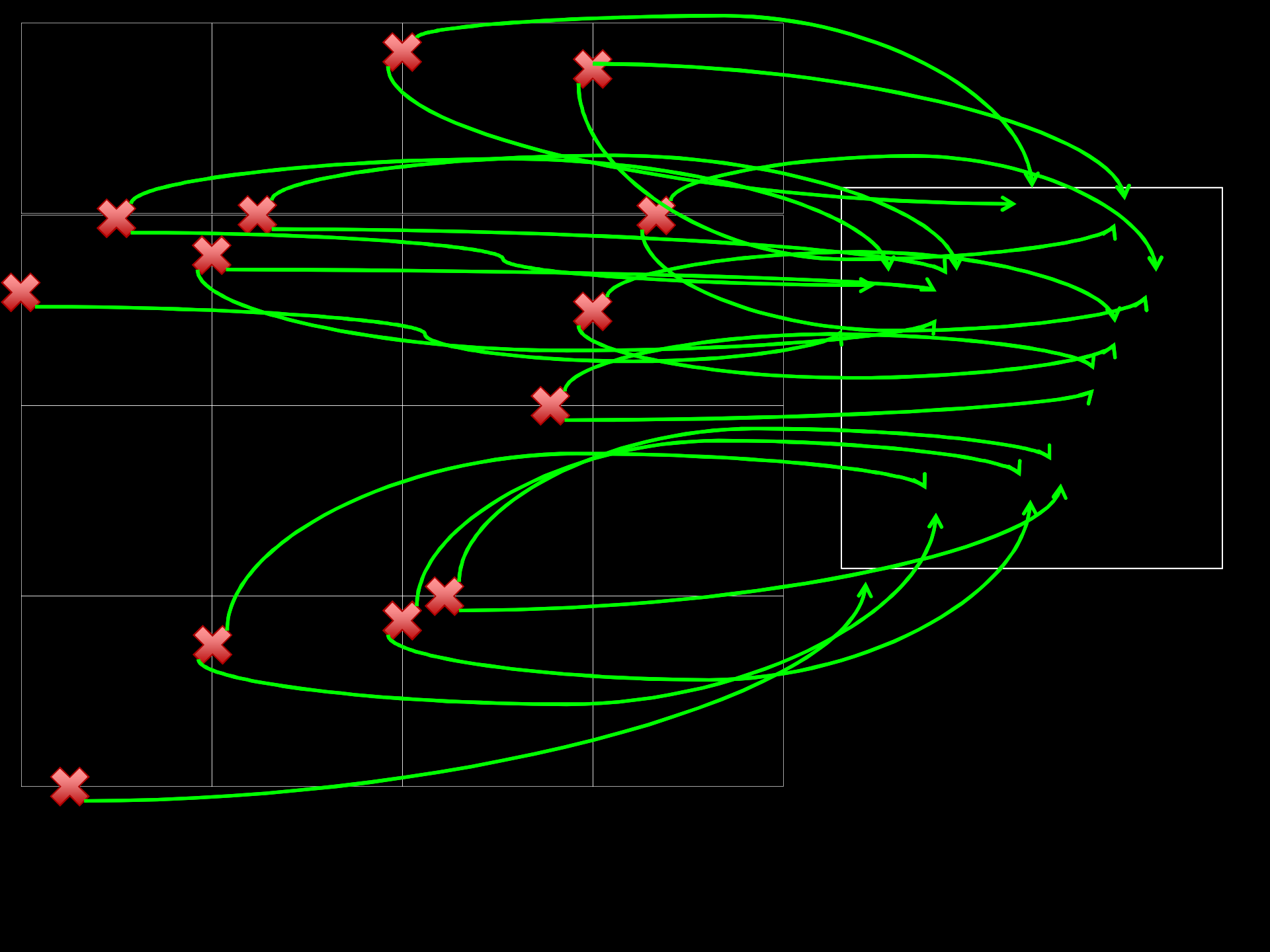


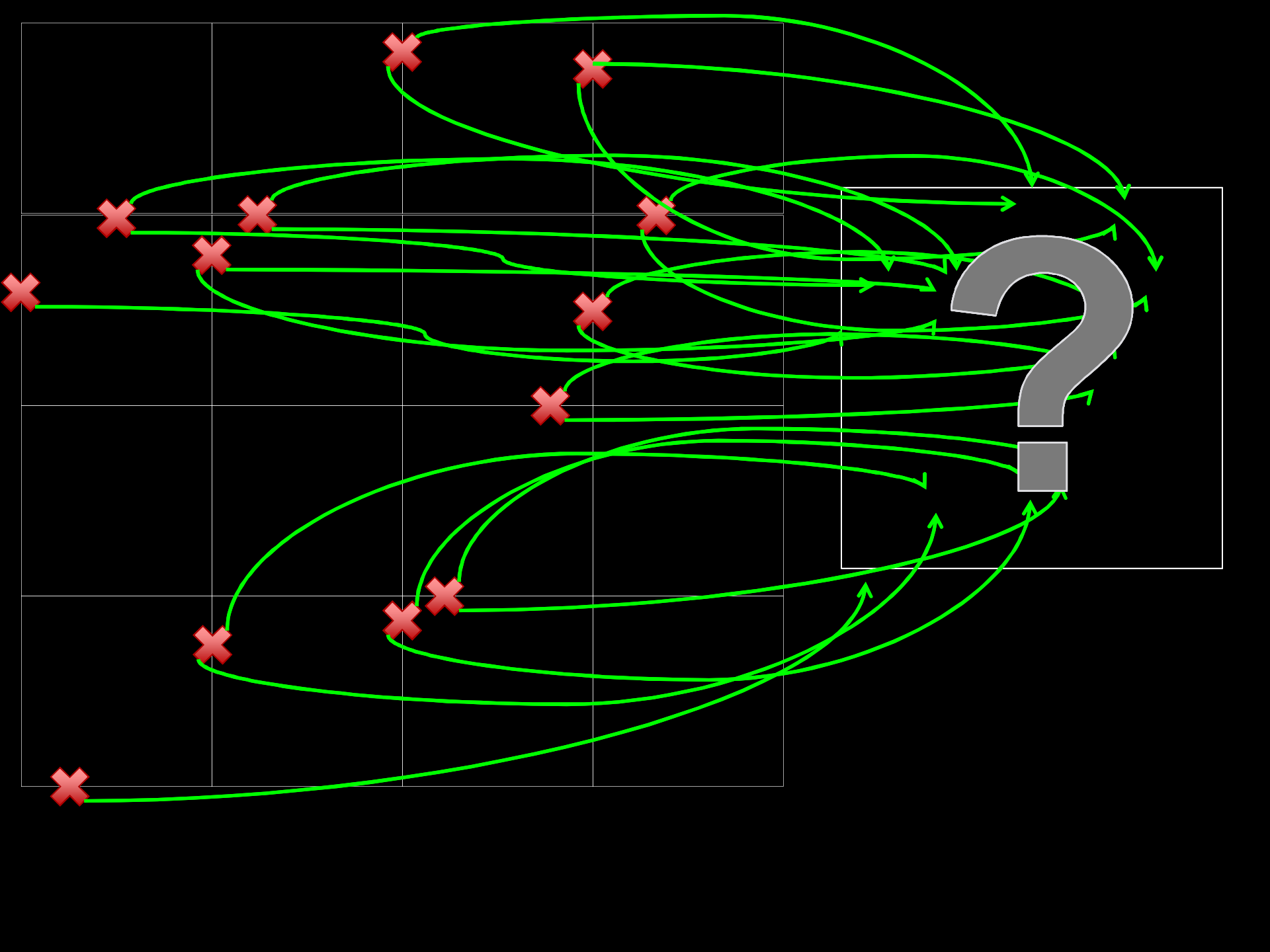


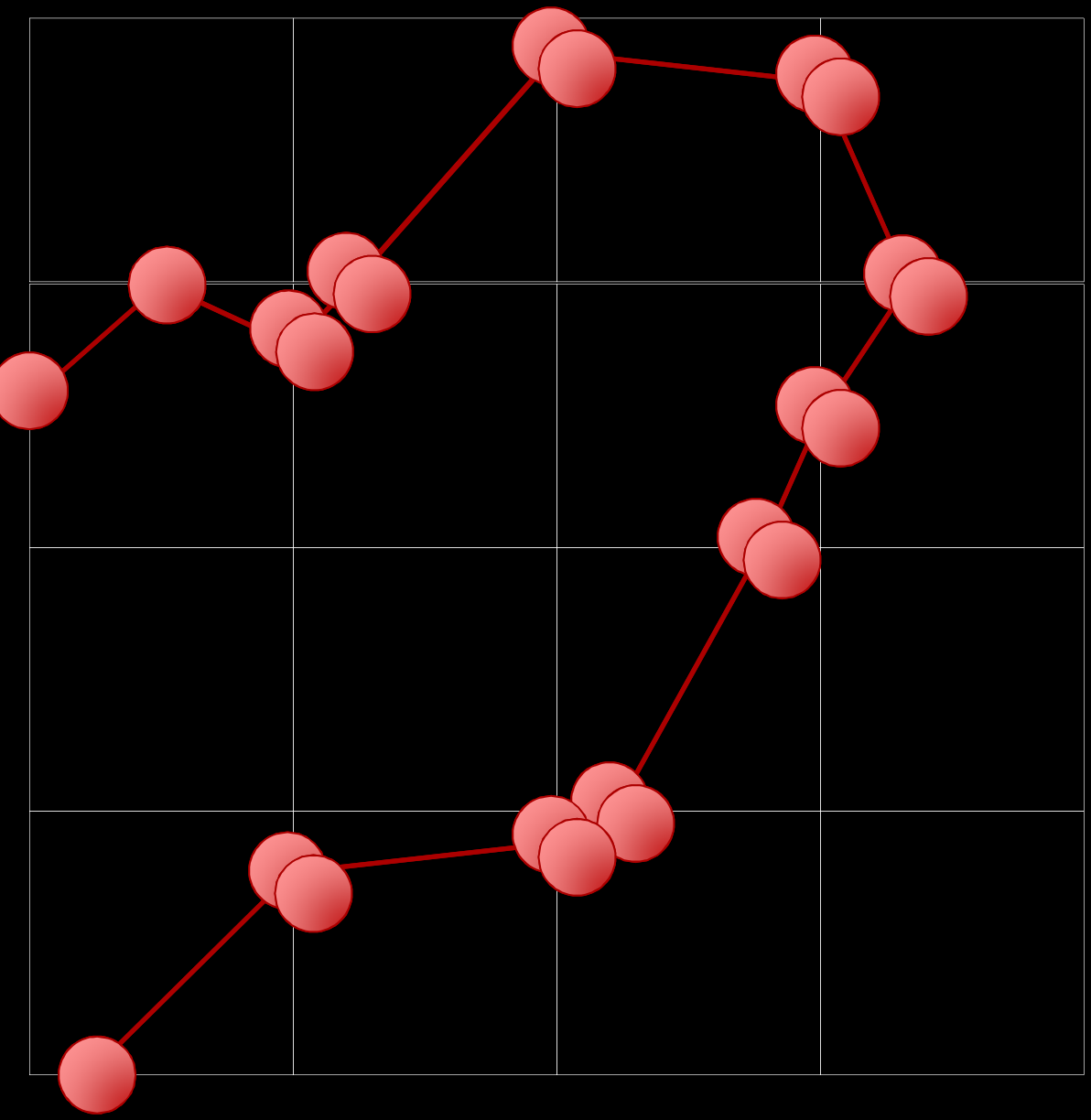






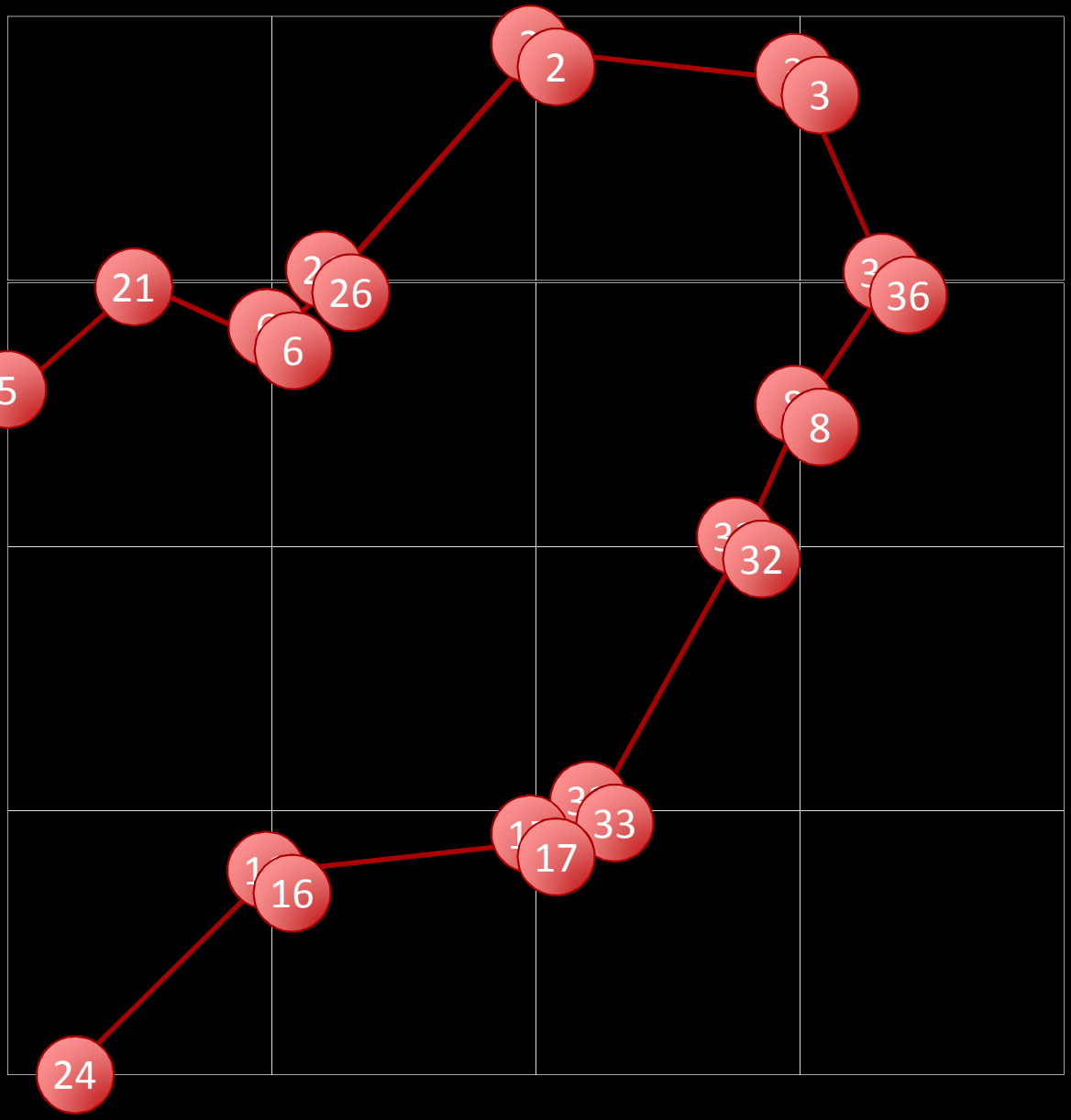


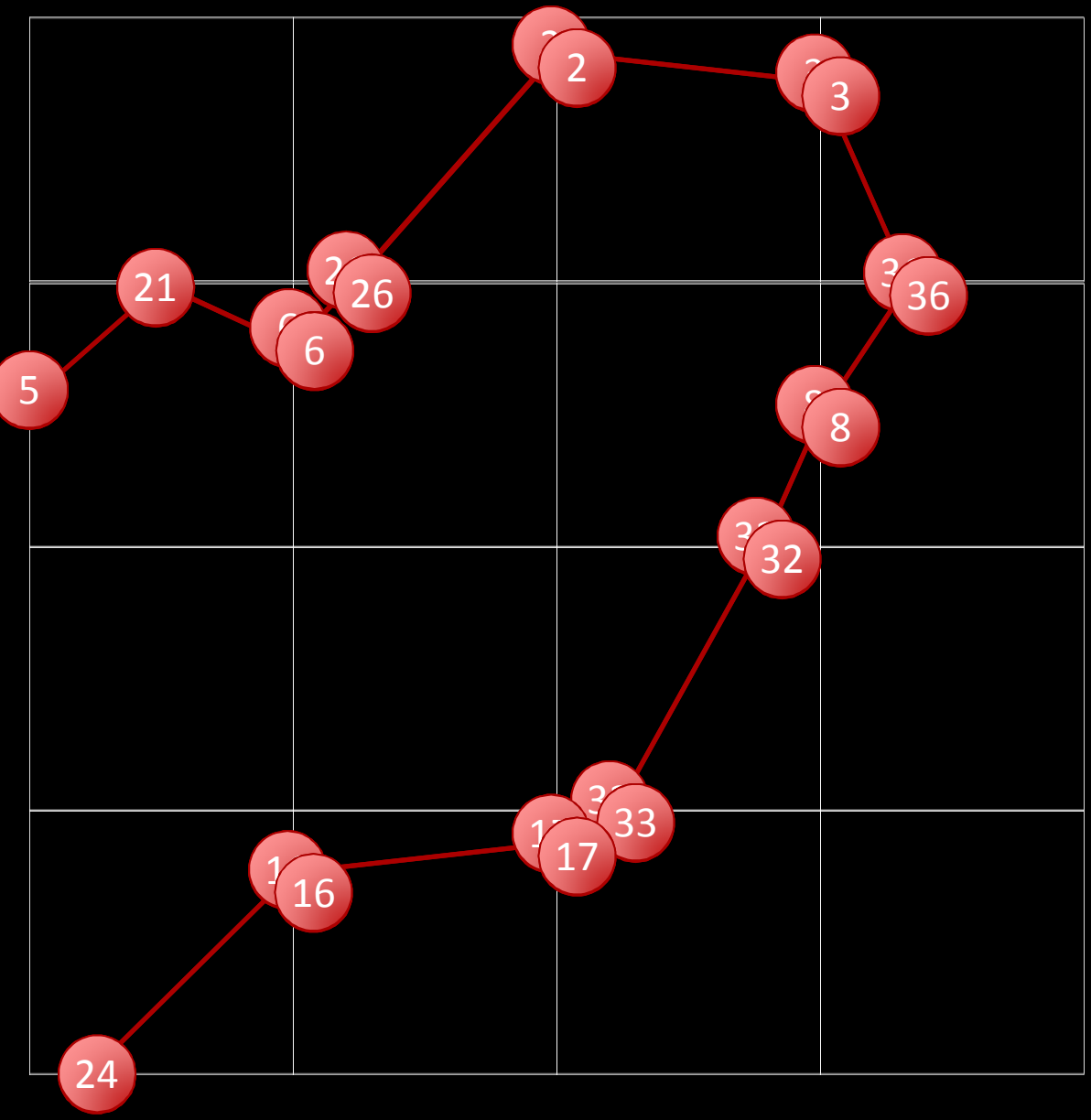


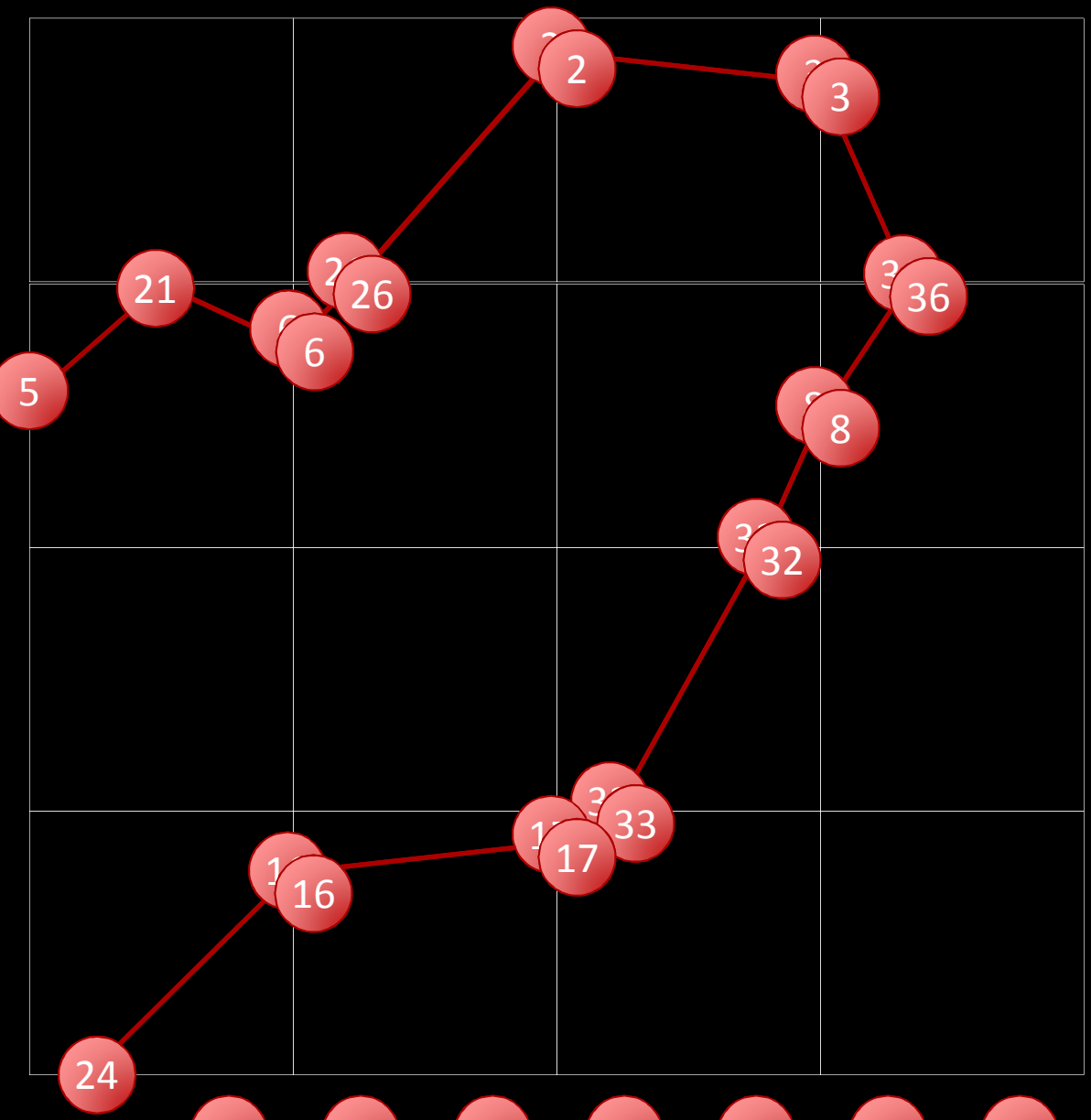


Faceted Normals

Smoothed Normals

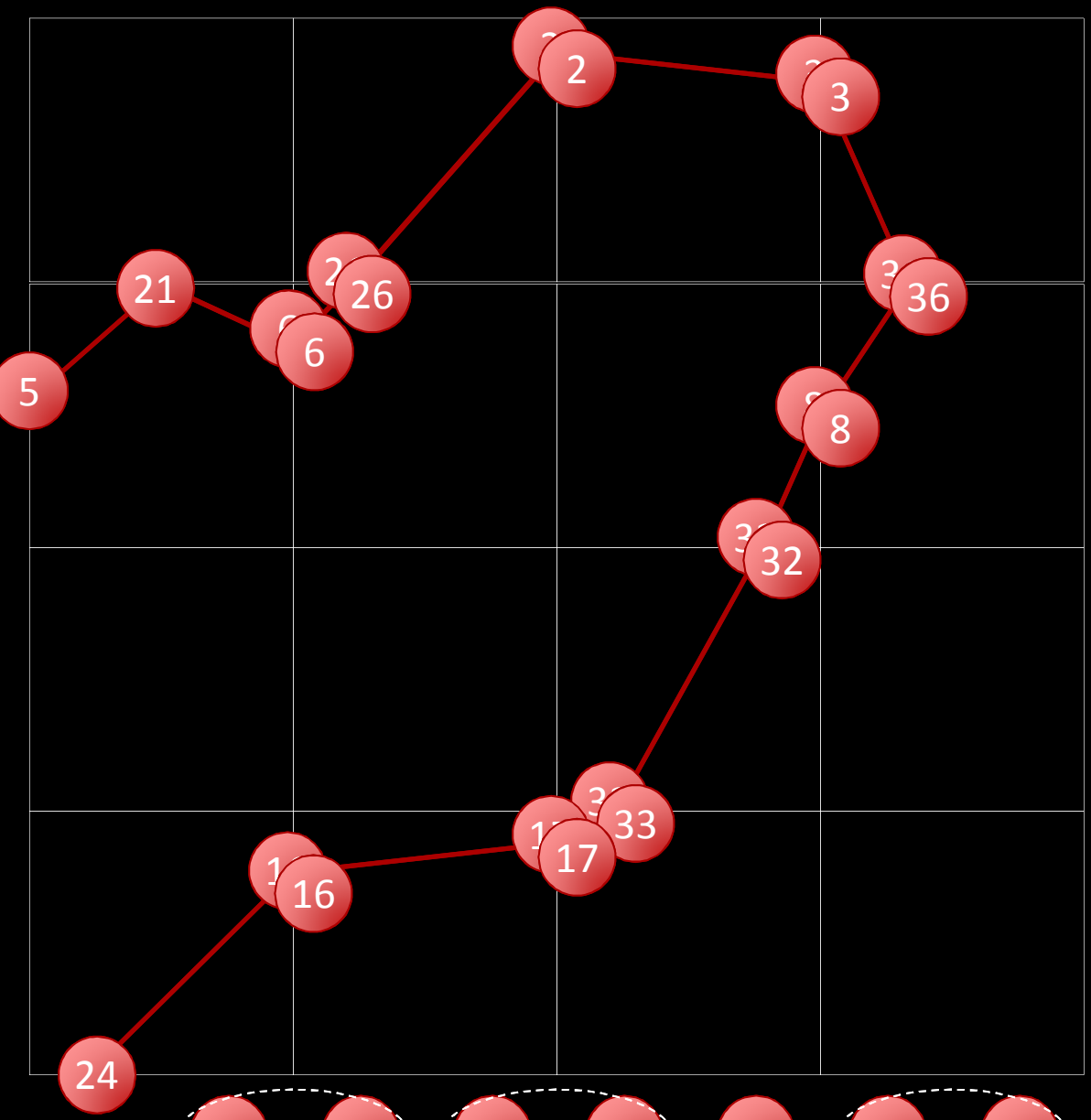

2
3
21
26
36
6
5
8
32
33
17
16
24


2
3
21
26
36
6
5
8
32
33
17
16
24



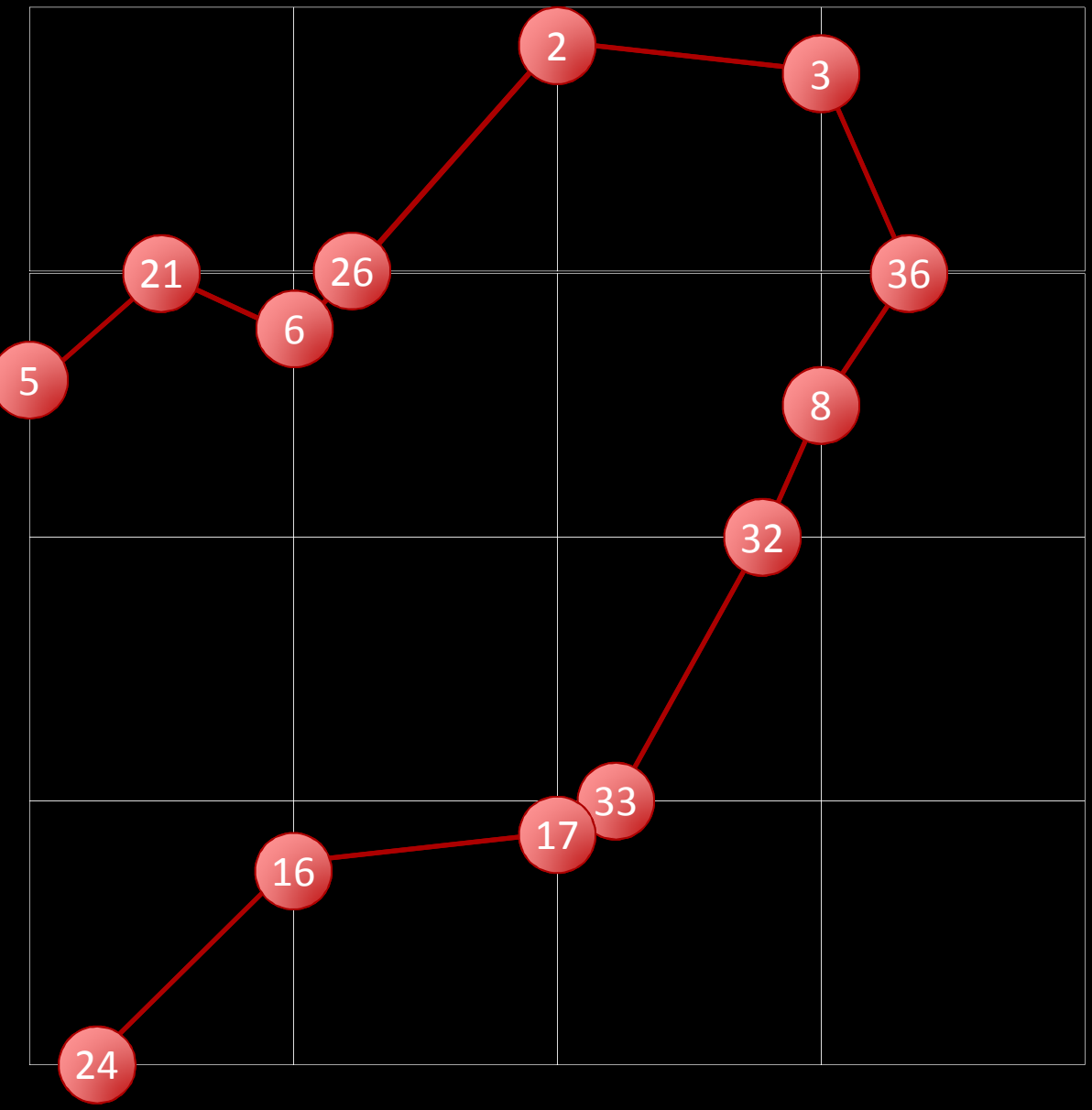
2 26 2 3 36 3 6 21 5 26 6

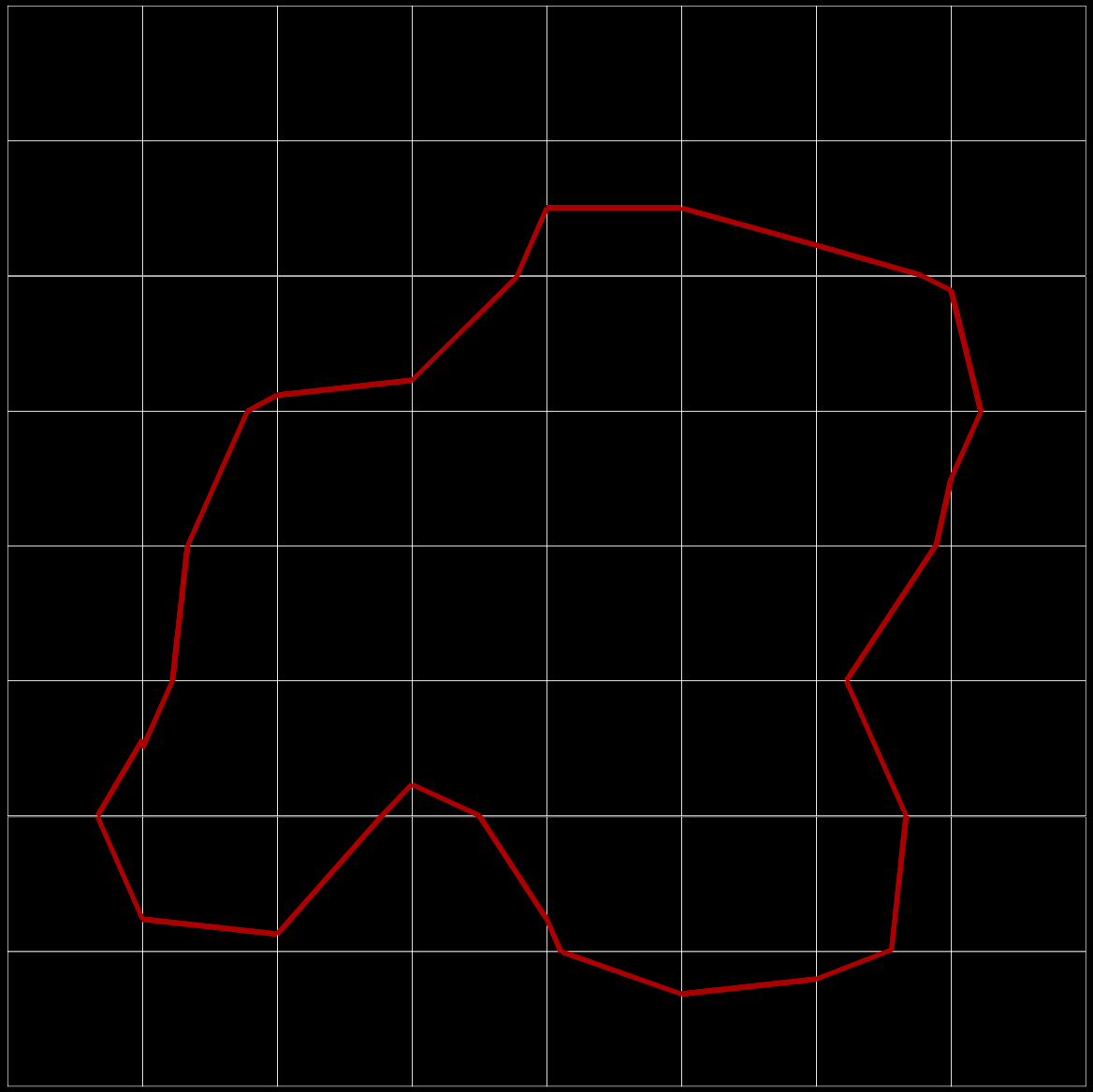
8 32 36 8 32 33 16 24 17 16 33 17



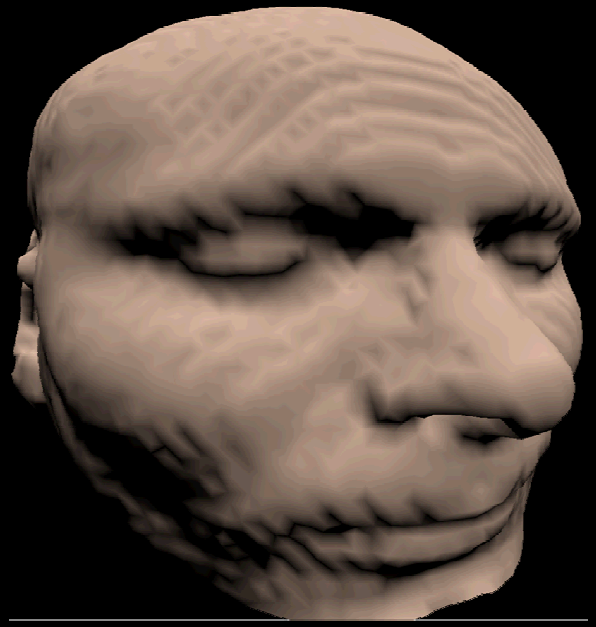
2 2 3 3 5 6 6 8 8 16 16

17 17 21 24 26 26 32 32 33 33 36 36


2
3
21
26
36
6
5
8
32
33
17
16
24



Faceted Normals



Smoothed Normals

Simplified Mesh

Project Goals (Summary)

Sandia National Laboratories

- Develop readiness for scientific data analysis and visualization at extreme scale.
  - In particular, address the challenges of emerging multi- and many-core architectures.
- Create a toolkit that well suited to design of visualization operations with a great number of shared memory threads.
- Develop a framework that adapts to emerging processor and compiler technologies.
- Design multi-purpose algorithms that can be applied to a variety of visualization operations.

Next Steps

- Converge on functional API
  - Alpha version released BSD (http://daxtoolkit.org)
- Unify algorithms within integrated toolkit
- Tackle real problems
  - Seeking early adopters interested in applying GPUs / accelerators to their problem domain