TempoEngine Documentation

Rostyslav Liapkin

# Conduction hear transfer between two grain squares

Sources are:

1. <https://bookdown.org/huckley/Physical_Processes_In_Ecosystems/heattransfer.html>
2. <https://www.sciencedirect.com/science/article/pii/S0370157323003770>

We are using a Fourier law of a heat conduction:

Where P – total heat transfer power, S – cross section area of the parallelepiped, ∆T – temperature difference between the faces, l – length of the parallelepiped, – thermal conductivity coefficient, h – “height” of our square (won’t be in the end formula).

As we are running in a 2D space, and every square has the same width, the length of the touch is always equal to the length of the parallelepiped (square in our case), the formula can be simplified to an equation

And it’s required to adjust the thermal conductivity coefficient for a 2D space to make simulation correct. So, it’s possible to calculate the change in temperature for two grain squares that are touching.

Let us have two squares with temperatures and , so the change in a temperature will be calculated as a

Where – thermal conductivity coefficient, t – time interval such that , in our simulation it will be time interval of engine update, smaller interval – more precise results, and – specific heat capacity for squares, and – mass of the square (depends only on density of the object, from material).

# Radiation transfer between two grain squares

Sources are:

1. <https://courses.lumenlearning.com/suny-physics/chapter/14-7-radiation/#:~:text=The%20rate%20of%20heat%20transfer%20by%20emitted%20radiation%20is%20determined,its%20absolute%20temperature%20in%20kelvin>.
2. <https://www.engineeringtoolbox.com/radiation-heat-transfer-d_431.html>

We are calculating radiation heat transfer both for losing heat to air due to radiation and transferring radiation between grain squares for now.

### Radiation heat transfer to air

If a hot object is radiating energy to its cooler surroundings the net radiation heat loss rate can be expressed with **Stefan-Boltzmann Law**as

where is emissivity coefficient of the object (1 - for a black body), is The Stefan-Boltzmann Constant, A – area of the emitting body, is hot body absolute temperature, is cold surrounding absolute temperature

So:

### Radiation heat transfer between objects

# Engine

## Objects

The only object that is available now in our simulation is GrainSquare, that is a basic object of our simulation. Inherited from the public abstract class EngineObject.

### EngineObject

Public abstract class implements interface **INotifyPropertyChanged**. EngineObject serves as the foundational class for objects in the simulation engine, providing common properties like position, temperature, size, and abstract methods that must be implemented

#### Properties

1. **Position** – position of the object, which point is said to be position is defined in inherited classes
2. **Size** – size of the object, defined for
3. **SimulationTemperature** - temperature of the object at the start of the simulation, won’t be changed during the run of the Engine
4. **CurrentTemperature** – temperature of the object at this moment. When engine is not running is equal to the **SimulationTemperature**
5. **Name** – name of the object, used as and ID for the object, there can’t be two objects that have the same name

#### Functions

1. **abstract public List<Polygon> GetPolygons()** - Returns polygons representing the object's shape. Must be implemented by subclasses.
2. **abstract public List<GrainSquare> GetSquares()** - Returns the object's squares. Must be implemented by subclasses.
3. **abstract public bool IsVisible(CanvasManager canvasManager)** - Determines if the object is visible on the given canvas. Must be implemented by subclasses.
4. **abstract public void GetObjectVisibleArea(out Vector2 topLeft, out Vector2 bottomRight)** - Gets the visible area of the object. Must be implemented by subclasses.
5. **abstract public void SetStartTemperature**() - Sets the starting temperature for the simulation. Must be implemented by subclasses.
6. **abstract public string GetObjectTypeString()** - Gets the type of the object as a string. Must be implemented by subclasses.
7. **abstract public ObjectType GetObjectType()** - Gets the type of the object as an ObjectType enum. Must be implemented by subclasses.
8. **abstract public string GetJsonRepresentation()** - Gets a JSON string representing the object's state. Must be implemented by subclasses.
9. **abstract public bool IsIntersecting(EngineObject obj)** - Determines if the object is intersecting with another object. Must be implemented by subclasses.
10. **abstract public void CacheProperties()** - Cache all the object's properties. Must be implemented by subclasses.
11. **abstract public List<GrainSquare> GetExternalSquares()** – Gets all external GrainSquare’s of an object, that can tranfer heat with other external GrainSquare’s of other objects

### GrainSquare

The GrainSquare class extends EngineObject and encapsulates the properties and behavior of square-shaped grain in the simulation, including thermal properties, position and selection state. It includes methods for rendering, visibility checks, serialization, etc.

#### Properties

1. Position – position of the left top corner of square, can be only integer, Point
2. Size – size of the square, always equal to the Engine.GridStep

#### Functions

1. public override List<Polygon> GetPolygons() – returns a list of GrainSquares, where the only square polygon is this GrainSquare, overrides the abstract method of EngineObject
2. private void SetCachedPoints() – caches the other 3 points of of the square to make all functions work faster
3. public override bool IsVisible(CanvasManager canvasManager) - determines whether any of the square's vertices are visible in the current view, overrides the abstract method of EngineObject
4. public override void GetObjectVisibleArea(out Vector2 topLeft, out Vector2 bottomRight)
5. public void AddEnergyDelta(double energyDelta) - Add energy to the grain square that was calculated in one simulation step
6. public void ApplyEnergyDelta() - Applies the energy delta to the grain square, updating the temperature.
7. public override void SetStartTemperature() - Sets the initial temperature of the grain to the simulation temperature, overrides the abstract method of EngineObject.
8. public override string GetObjectTypeString() - Provides the type identifier for Grainsquare objects as string, overrides the abstract method of EngineObject.