ThedyxEngine Documentation

Rostyslav Liapkin

User manual

## General description

#### MacOS

A screenshot of a computer

AI-generated content may be incorrect.

Windows

A screenshot of a computer

AI-generated content may be incorrect.

ThedyxEngine is a 2D physics engine designed to simulate heat transfer across different materials using a visually intuitive approach. The engine supports various forms of heat transfer mechanisms including conduction, convection, and radiation, presenting them in a visually engaging manner that changes color based on the temperature of the objects.

Features Conduction: Simulate heat transfer through direct contact. Convection: Model the transfer of heat through fluids and gases. Radiation: Represent the emission of heat through electromagnetic waves. Visual Representation: Visualize temperature changes using color gradients to represent varying intensities of heat. How It Works Shape Division Objects are divided into a grid of small squares, allowing for detailed and localized temperature calculations. This division aids in accurately simulating how heat diffuses through different materials.

Heat Transfer Calculations The engine calculates heat transfer between adjacent squares by considering factors such as temperature differences, the thermal conductivity of the material, and the simulation timestep.

Temperature Updates Temperatures of individual squares are updated based on net heat gain or loss, integrating effects from conduction and radiation.

## Controls

### UI components

App is made from three components

1. Canvas
2. Properties tab
3. UI bar
4. Objects list

#### Canvas

Graphic interpretation of the simulation, that displays objects colored by their temperature. You can move on your scene by Keys “WASD” and zoom by mouse scroll

##### Color interpretation

1. - violet
2. - blue
3. - light blue
4. - green
5. - yellow
6. - orange
7. - red

### Properties tab

Tab where properties of the object are changed. It’s forbidden to change objects parameters during the run of the simulation.

##### Object properties

1. Name – unique ID of the object, there can’t be two objects with the same name.
2. Position – position of the object, for example, for square it’s a left top corner. There can’t be two objects that are at the same position or have intersection.
3. Size – size of the object.
4. - current temperature of the object
5. Material – material of the object, each material has its unique parameters, at this moment there is available only one material.

### Objects list

Component of the UI, where a list of all objects is shown, you can select any object to change its properties on the tab and to show its position on the Canvas.

To move and zoom canvas to a needed object, double-click on the needed object in the list. To delete object, select it and press “Delete”. To change selection of the object you can also move by the arrows “Up” and “Down” or use “W” and “S”.

### UI bar

UI bar contains two parts – general simulation control and simulation run controls.

1. A computer screen shot of a computer

   Description automatically generated**Add Button** – add objects to the simulation.
2. **Save Button** – save simulation to the file
3. **Open Button** – open simulation from the file
4. **Delete Button** – delete all the objects from the scene



1. **Start Button** – start simulation
2. **Pause Button** – pause the simulation
3. **Stop Button** – stop the simulation
4. **Reset Button** – reset the simulation
5. **Simulation time Label** – current time of the simulation

Technical documentation

# 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>
3. *https://web.mit.edu/16.unified/www/FALL/thermodynamics/notes/node136.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:

where, k is count of a side of a square that are not touching other squares

In our simulation to simplify calculations we assume that every object loses radiation heat to air if it’s not touching other squares. So, we are adjusting area based on how many “free” sides do we have.

### Radiation heat transfer between objects

If radiation heat is transferred between two objects, it can be expressed with this formula:

where and are emissivity coefficients of the objects, and are determine surface areas(they are always equal), and are the temperatures of the objects, is The Stefan-Boltzmann Constant, is a view factor between objects

So,

Result:

To simplify calculations, we calculate radiation between every two objects in our simulation and we assume that the view factor is always 1. It will be calculated as:

# Engine

## Engine calculations

For now, all calculations are made in one thread, for every step of the simulation (that equals for now to 1/120 sec).

Engine includes two managers: **ConductionTransferManager** and **RadiationHeatManager**, where heat transfers are being calculated.

Now Engine optimized to perform conduction transfer with time , where n is count of objects and radiation transfer in time . Radiation transfer will be remade after.

## OptimizationManager

The **OptimizationManager** class provides methods for optimizing the engine by setting adjacent squares to be touching. We don’t have to check if objects are touching on every time step, so it will be done before the simulation starts. Optimization has a big enough time complexity, but it will be done only once before the start of the engine.

### Functions

1. public static void Optimize(List<EngineObject> objects) – main function of the class, that clears optimized data and recalculates it. Called only after some changes are made in the simulation.
2. private static void FillExternalSquares(List<GrainSquare> squares1, List<GrainSquare> squares2) - fill adjuscent squares for two lists of squares
3. private static void OptimizeTouching(List<EngineObject> objects) - optimize touching objects, by setting adjuscent squares for every square of an object
4. public static void ClearOptimization(List<EngineObject> objects) - clear optimization

## RadiationTransferManager

The **RadiationTransferManager** provides methods for calculating and transferring radiation heat between objects in the simulation. It includes methods for transferring radiation heat between two objects and radiation heat loss to air.

### Functions

1. public static void TransferRadiationHeat(List<EngineObject> objects) – main method of the class, provides transferring radiation heat loss to air, then transferring radiation heat between objects.
2. private static void TransferRadiationHeatLooseToAir(EngineObject obj) – transfers radiation heat loss to air
3. private static void TransferRadiationBetweenTwoObjects(EngineObject obj1, EngineObject obj2) – transfer radiation heat between two objects

## ConductionTransferManager

The ConductionTransferManager class provides methods for calculating and transferring conduction heat between objects in the simulation,

### Functions

1. public static void TransferConductionHeat(List<EngineObject> objects) – main method of the class, transfers heat between all GrainSquares of the objects.
2. private static void TransferHeatForObject(EngineObject obj) – transfers heat for every GrainSquare with its adjacent squares.
3. private static void TranferHeatBetweenTwoSquares(GrainSquare sq1, GrainSquare sq2) – transfers heat between two adjacent squares based on calculated formulae.

## 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 an ID for the object, there can’t be two objects that have the same name
6. **Material** – material of the object, that includes emissivity, density, specific heat capacity

#### 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.