TEXAS A&M UNIVERSITY

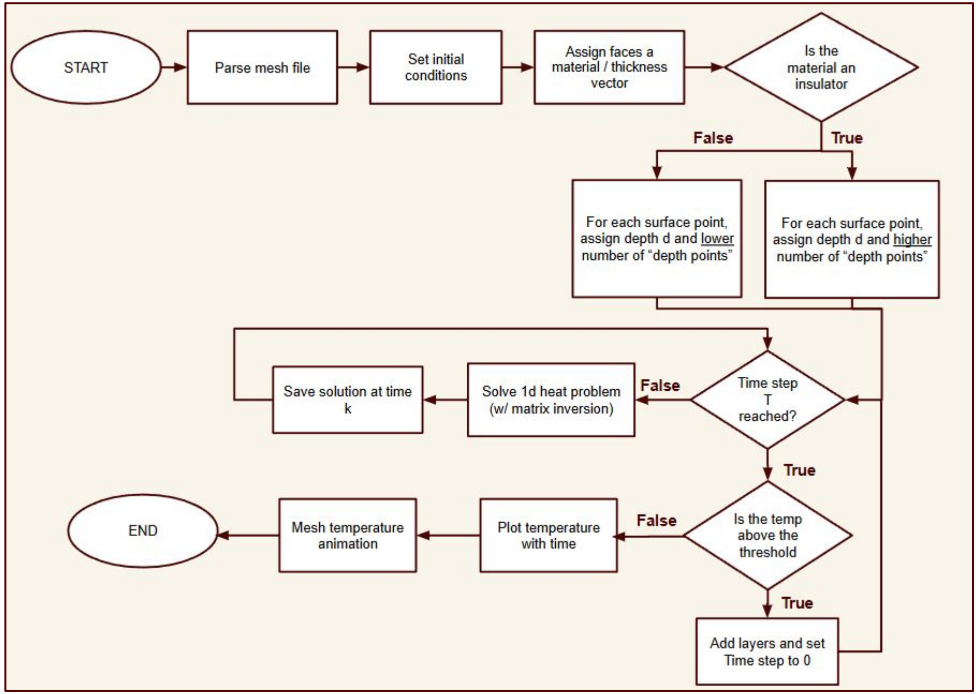
Thermal Protection Solver Final Report

Date: 05/01/2025

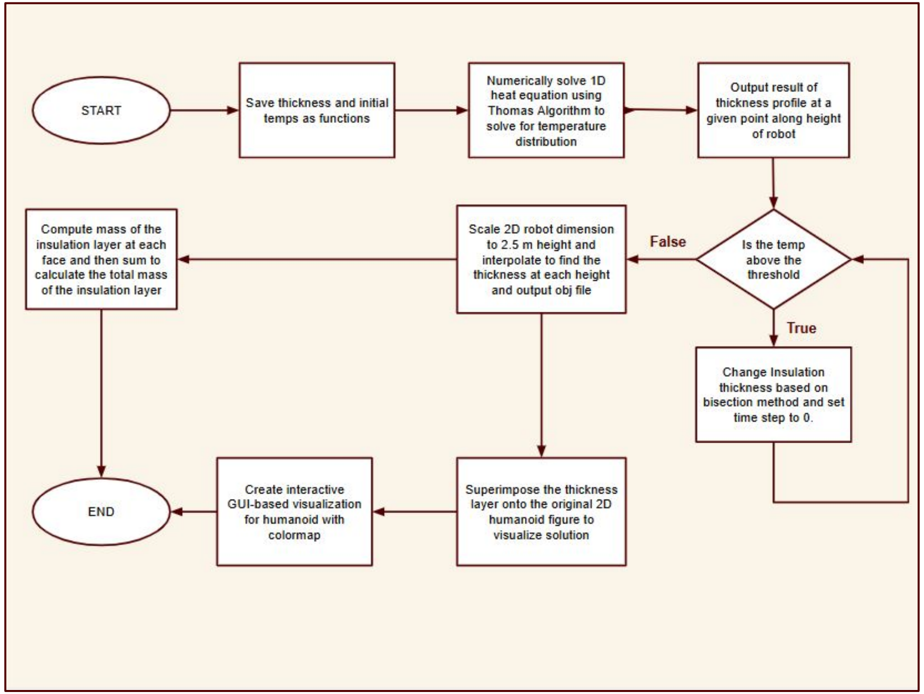
Group Members: Nishi Mishra, Layan Samandar, Parina Patel, Hassan Niaz, Bihao Zhang, Naveen Jagadeesan

Flow Diagrams

Initial:



Final:



Matrix Inversion

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

$$\frac{u_{i,n+1} - u_{i,n}}{\Delta t} - \alpha^2 \frac{u_{i+1,n+1} - 2u_{i,n+1} + u_{i-1,n+1}}{(\Delta x)^2} \approx 0$$

$$u_{i,n} \approx u_{i,n+1} - \alpha^2 \Delta t \cdot \frac{u_{i+1,n+1} - 2u_{i,n+1} + u_{i-1,n+1}}{(\Delta x)^2}$$

$$u_{i,n} \approx u_{i,n+1} \left(1 + \frac{2\alpha^2 \Delta t}{(\Delta x)^2}\right) - \frac{\alpha^2 \Delta t}{(\Delta x)^2}(u_{i+1,n+1} + u_{i-1,n+1})$$

$$\boxed{\lambda = \frac{\alpha^2 \Delta t}{(\Delta x)^2}}$$

$$u_{i,n} \approx u_{i,n+1}(1 + 2\lambda) - \lambda(u_{i+1,n+1} + u_{i-1,n+1})$$

Matrix Inversion - Continued

In compact form:

$$\mathbf{u}_n \approx A\mathbf{u}_{n+1}$$

where:

$$A = \begin{bmatrix} 1+2\lambda & -\lambda & 0 & \cdots & 0 \\ -\lambda & 1+2\lambda & -\lambda & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -\lambda & 1+2\lambda & -\lambda \\ 0 & \cdots & 0 & -\lambda & 1+2\lambda \end{bmatrix}$$

$$\mathbf{u}_{n+1} = A^{-1}\mathbf{u}_n$$

So, The heat equation solution boils down to solving Ax=b. Since A has a special structure, i.e., A is tridiagonal, we'll be using **Thomas Algorithm** for the solution

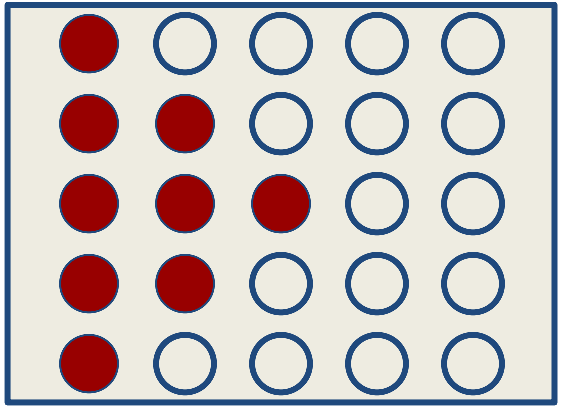
Tridiagonal Matrices

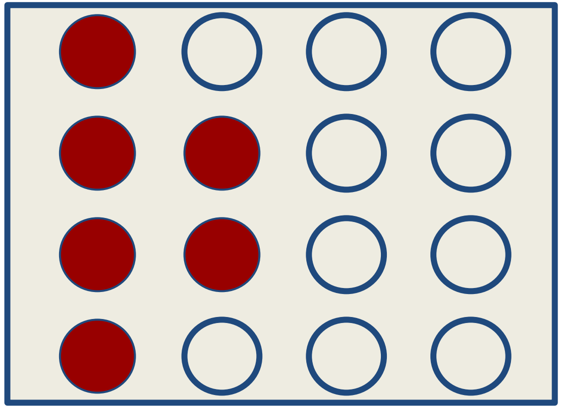
For Tridiagonal systems, an observation is that their inverse has a special structure:

```
A =

    22     1     0     0
     1    22     1     0
     0     1    22     1
     0     0     1    22

>> inv(A)

ans =

    0.0455   -0.0021    0.0001   -0.0000
   -0.0021    0.0456   -0.0021    0.0001
    0.0001   -0.0021    0.0456   -0.0021
   -0.0000    0.0001   -0.0021    0.0455
```

Tridiagonal Matrices

For Tridiagonal systems, an observation is that their inverse has a special structure:





For even number of columns: $n(n+2)/4$

For odd number of columns: $(n+1)^2/4$

Why Thomas Algorithm

- Specialized form of Gaussian elimination tailored for tridiagonal matrices

- Solves the inverse of an nxn matrix in O(n) time as opposed to standard inverse algorithms which take O(n^3) time

- Has low memory head where instead of storing the whole matrix (n^2 terms), we only need to store three columns (3n-2 terms)

Why Thomas Algorithm

TEXAS A&M UNIVERSITY

Command Window

```
Thomas Algo Finished in 0.0042196000 s
MATLAB's Inverse Finished in 0.3688809000 s

difference =

   1.0e-13 *

    0.4529
   -0.0297
   -0.5420
    0.0715
    0.6464

>>
```

87 times faster in this case

Generally, 50-100x faster

Thermal Simulation Assumptions

External boundary (robot surface):

- Constant temperature from initialTemp(z) function
  - 900 K at toes

Innermost boundary:

- Assumed to be in contact with still air modeled as an
  - insulator → zero heat flux

Interface Handling:

- Assume heat flux to be continuous across material boundaries

Interface Handling

- We assume heat flux is continuous between materials:

$$k_1 \frac{\partial T}{\partial x} = k_2 \frac{\partial T}{\partial x}$$

- We used thermal resistance to enforce this:

$$R = \frac{\Delta x}{k}$$

$$T_{\text{interface}} = \frac{T_L R_2 + T_R R_1}{R_1 + R_2}$$

- This guarantees the flux entering neighboring elements equals flux leaving

Thermal Resistance & Harmonic Mean Equivalence

Thermal resistances **add in series**:

$$R_{\text{total}} = \frac{\Delta x_1}{k_1} + \frac{\Delta x_2}{k_2}$$

So for equal spacing ($\Delta x_1 = \Delta x_2 = \Delta x/2$):

$$R_{\text{total}} = \frac{\Delta x}{2k_1} + \frac{\Delta x}{2k_2} = \frac{\Delta x}{2}\left(\frac{1}{k_1} + \frac{1}{k_2}\right)$$

The **effective conductivity** is defined by:

$$R_{\text{total}} = \frac{\Delta x}{k_{\text{eff}}} \Rightarrow \frac{\Delta x}{k_{\text{eff}}} = \frac{\Delta x}{2}\left(\frac{1}{k_1} + \frac{1}{k_2}\right)$$

Cancel Δx:

$$\boxed{\frac{1}{k_{\text{eff}}} = \frac{1}{2}\left(\frac{1}{k_1} + \frac{1}{k_2}\right) \Rightarrow k_{\text{eff}} = \frac{2k_1k_2}{k_1 + k_2}}$$

Evolution of the Heat Solver

TEXAS A&M UNIVERSITY

Previous Approaches:

1 Insulation Only

- Simulated just thermal protection layer
- Only interface temp stayed above carbon fiber's glass transition temp (350K)

2 Thermal Resistance + Inertia

$$\frac{\partial T}{\partial t} = \frac{1}{\rho c_p} \frac{\partial}{\partial x} \left(k \frac{\partial T}{\partial x} \right)$$

- Added transient term using ρ·cp (thermal inertia)
- Used half-cell R = Δx / k and BTCS with varying properties

3 Sequential Layer Solver

- Solved each layer one after another for part of dt
- Assumed constant outer temp + insulated inner face
- Layers temp calcs evolved sequentially over time

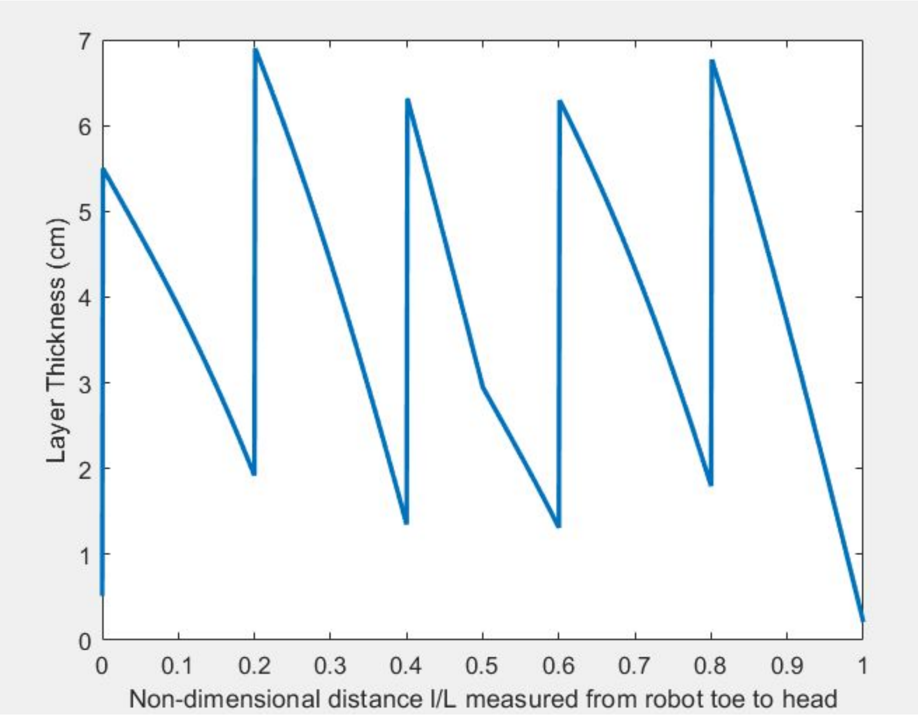
Final Method:

Harmonic Mean Solver

- Uses harmonic mean of k at interfaces
  - equivalent to thermal resistance
- Based on steady-state conduction flux, applied over time
- Solves full stack with BTCS, synchronized in time

Validation

```
z=0.00 units -> T_CF=339.43K, T_GLUE=339.43K, T_STL=339.43K
z=0.04 units -> T_CF=317.55K, T_GLUE=317.53K, T_STL=317.05K
z=0.08 units -> T_CF=318.31K, T_GLUE=318.30K, T_STL=318.01K
z=0.12 units -> T_CF=319.53K, T_GLUE=319.52K, T_STL=319.38K
z=0.16 units -> T_CF=321.39K, T_GLUE=321.38K, T_STL=321.34K
z=0.20 units -> T_CF=324.30K, T_GLUE=324.30K, T_STL=324.30K
z=0.24 units -> T_CF=313.26K, T_GLUE=313.25K, T_STL=312.88K
z=0.28 units -> T_CF=314.58K, T_GLUE=314.57K, T_STL=314.34K
z=0.36 units -> T_CF=319.65K, T_GLUE=319.65K, T_STL=319.61K
z=0.40 units -> T_CF=324.71K, T_GLUE=324.71K, T_STL=324.71K
z=0.44 units -> T_CF=313.55K, T_GLUE=313.54K, T_STL=313.16K
z=0.48 units -> T_CF=315.98K, T_GLUE=315.97K, T_STL=315.72K
z=0.52 units -> T_CF=318.44K, T_GLUE=318.44K, T_STL=318.31K
z=0.56 units -> T_CF=320.40K, T_GLUE=320.40K, T_STL=320.36K
z=0.60 units -> T_CF=323.11K, T_GLUE=323.11K, T_STL=323.11K
z=0.64 units -> T_CF=311.81K, T_GLUE=311.80K, T_STL=311.47K
z=0.68 units -> T_CF=312.81K, T_GLUE=312.80K, T_STL=312.60K
z=0.72 units -> T_CF=314.24K, T_GLUE=314.23K, T_STL=314.13K
z=0.76 units -> T_CF=316.37K, T_GLUE=316.37K, T_STL=316.34K
z=0.80 units -> T_CF=309.93K, T_GLUE=309.92K, T_STL=309.50K
z=0.84 units -> T_CF=311.09K, T_GLUE=311.08K, T_STL=310.77K
z=0.88 units -> T_CF=312.79K, T_GLUE=312.79K, T_STL=312.59K
z=0.92 units -> T_CF=315.37K, T_GLUE=315.36K, T_STL=315.25K
z=0.96 units -> T_CF=319.52K, T_GLUE=319.52K, T_STL=319.48K
z=1.00 units -> T_CF=327.10K, T_GLUE=327.10K, T_STL=-0.00K
```



Our Solving Functions

```cpp
vector<vector<double>> solveMultiLayer(
    const vector<double>& layerThickness,
    const vector<MaterialProperties>& materials,
    double missionDuration,
    double dt,
    double dx,
    double T_ext,
    bool dummy
)
{
    double totalThickness = 0;
    for (double t_cm : layerThickness) totalThickness += t_cm/100.0;
    int N = static_cast<int>(totalThickness / dx) + 1;
    int timePoints = static_cast<int>(missionDuration / dt) + 1;

    vector<double> T(N, 300.0); // Initialize temperatures
    vector<vector<double>> T_out(timePoints, vector<double>(N, 300.0));
    vector<double> alpha(N);    // thermal diffusivity at each node

    int idx = 0;
    for (size_t L = 0; L < materials.size(); ++L) {
        double t_m = layerThickness[L]/100.0;
        int nL = static_cast<int>(t_m / dx);
        for (int i = 0; i < nL && idx < N; ++i, ++idx) {
            alpha[idx] = materials[L].thermalConductivity / (materials[L].density * materials[L].specificHeatCapacity);
        }
    }
    while (idx < N) {
        alpha[idx++] = materials.back().thermalConductivity / (materials.back().density * materials.back().specificHeatCapacity);
    }

    vector<double> a(N-1, 0.0), b(N, 0.0), c(N-1, 0.0);

    for (int i = 0; i < N; ++i) {
        if (i == 0) {
            b[i] = 1.0;
            c[i] = 0.0;
        }
        else if (i == N-1) {
            double alphaL = alpha[i-1];
            double alphaR = alpha[i];
            double alpha_eff = 2.0 * alphaL * alphaR / (alphaL + alphaR);

            double lambda = alpha_eff * dt / (dx * dx);
            a[i-1] = -lambda;
            b[i] = 1.0 + lambda;
        }
        else {
            double alphaL = alpha[i-1];
            double alphaC = alpha[i];
            double alphaR = alpha[i+1];

            double alpha_eff_L = 2.0 * alphaL * alphaC / (alphaL + alphaC);
            double alpha_eff_R = 2.0 * alphaC * alphaR / (alphaC + alphaR);

            double lambda_L = alpha_eff_L * dt / (dx * dx);
            double lambda_R = alpha_eff_R * dt / (dx * dx);

            a[i-1] = -lambda_L;
            b[i]   = 1.0 + lambda_L + lambda_R;
            c[i]   = -lambda_R;
        }
    }

    T[0] = T_ext;
    for (int i = 0; i < timePoints; i++) {
        vector<double> d = T;
        d[0] = T_ext;
        T = thomas_algorithm(a, b, c, d);
        T_out[i] = T;  // Safe: using integer loop index
    }

    return T_out;
}
```

calculateRequiredInsulationThicknessMultiLayer()

MakeTimeSolution()

```cpp
// New function: Calculate required insulation thickness by solving full multilayer ODE stack
vector<double> calculateRequiredInsulationThicknessMultiLayer(double missionDuration, double dz, double dt, bool normalized)
{
    double totalHeight = 2.50;              // robot height in meters
    int Nz = static_cast<int>(totalHeight / dz) + 1;
    vector<double> insThick(Nz), zVals(Nz);
    vector<MaterialProperties> mats = { THERMAL_PROTECTION, CARBON_FIBER, GLUE, STEEL };
    vector<double> zValues((int)(totalHeight/dz)+1, 0.0);

    for (int i = 0; i < Nz; ++i) {
        double z = i * dz;
        zVals[i] = z;
        double dx = 0.0001;  // Spatial step size

        // fixed layer thicknesses (cm)
        double cf = carbonThickness(z);
        double gl = glueThickness(z);
        double st = steelThickness(z);

        // binary search bounds (cm)
        double lo = 0.1, hi = 50.0;
        double tol = 0.01;
        double mid = 0;

        while (hi - lo > tol) {
            mid = 0.5 * (lo + hi);
            vector<double> layers = { mid, cf, gl, st };
            double T_ext = initialTemp(z);

            // solve full stack
            auto Tdist = solveMultiLayer(
                layers, mats,
                missionDuration,
                dt, dx,
                T_ext,
                true
            );

            // index at insulation/carbon interface
            int idx_if = static_cast<int>((mid/100.0) / dx + 0.5);
            double T_if = Tdist.back()[idx_if];

            // check against carbon fiber limit
            if (T_if <= CARBON_FIBER.glassTransitionTemp-3) {
                hi = mid;
            } else {
                lo = mid;
            }
        }

        insThick[i] = 0.5 * (lo + hi);
        if(normalized)
        {
            cout << "z=" << normalizePosition(z) << " units, ins_thick=" << insThick[i]
                 << " cm -> T_if=";
        }
        else
        {
            cout << "z=" << z << " m, ins_thick=" << insThick[i]
                 << " cm -> T_if=";
        }
        // final solve for reporting
        auto Tfin = solveMultiLayer({insThick[i], cf, gl, st}, mats,
                                    missionDuration, dt, dx, initialTemp(z), normalized);
        int idx = static_cast<int>((insThick[i]/100.0) / dx + 0.5);
        cout << Tfin.back()[idx] << " K\n";
        zValues[i] = z;
    }

    // outputting thickness values in a csv
    string filename = "Thickness_" + to_string((int)(missionDuration/3600))+"_hr.csv";
    writeVectorsToCSV(filename, zValues, insThick);

    std::cout << "Output written in a csv!" << std::endl;
    return insThick;
}
```

OBJ Scaling and Thickness Parsing

TEXAS A&M UNIVERSITY

- Scale the 2D robot dimensions to the 2.5 m height
- Interpolate to find the thickness at each height
  - Uses csv with height vs thickness
- Optimization: Error handling for input obj

	A	B
1	z value	thickness (cm)
2	0	21.7333
3	0.1	8.18621
4	0.2	9.30092
5	0.3	10.9029
24	2.2	7.10805
25	2.3	9.16082
26	2.4	12.5963
27	2.5	20.5089

```
# Blender v2.71 (sub 0) OBJ File: 'Man.blend'
# www.blender.org
mtllib Man.mtl
o Human_Cylinder
v 0.271502 3.129741 0.0
v 0.341832 4.004711 0.0
v 0.417949 3.129741 0.0
v 0.488278 4.004711 0.0
v 0.271502 3.129741 0.0
```

Original 2D OBJ file

```
v 0.128951 1.48648 0.171449
v 0.162354 1.90205 0.114159
v 0.198507 1.48648 0.171449
v 0.23191 1.90205 0.114159
v 0.128951 1.48648 0.171449
```

Updated OBJ file

```
// Generate scaled/thickened vertices
for (const auto& v : original_vertices) {
    // Normalize y to [0, 2.5] meters
    double normalized_y_m = ((v.y - y_min) / (y_max - y_min)) * 2.5;
    double t_cm = InterpolateThickness(normalized_y_m, thickness_data);
    double t_m = t_cm / 100.0;  // Convert to meters

    // Scale x and y; apply thickness to z
    Vertex front = {v.x * scale, v.y * scale, v.z + t_m};
    Vertex back  = {v.x * scale, v.y * scale, v.z};

    front_vertices.push_back(front);
    back_vertices.push_back(back);
}
```

```
// ---------- Interpolate Thickness ----------
// Given a y-position, return the interpolated thickness from the CSV data
double InterpolateThickness(double y, const std::vector<ThicknessEntry>& data) {
    if (data.empty()) return 0.0;

    // Clamp to the bounds
    if (y <= data.front().y_m) return data.front().thickness_cm;
    if (y >= data.back().y_m) return data.back().thickness_cm;

    // Linear interpolation
    for (size_t i = 0; i < data.size() - 1; ++i) {
        if (y >= data[i].y_m && y <= data[i + 1].y_m) {
            double y0 = data[i].y_m;
            double t0 = data[i].thickness_cm;
            double y1 = data[i + 1].y_m;
            double t1 = data[i + 1].thickness_cm;
            return t0 + (t1 - t0) * (y - y0) / (y1 - y0);
        }
    }
    return data.back().thickness_cm;
}
```

Thermal Insulation Layer Mass

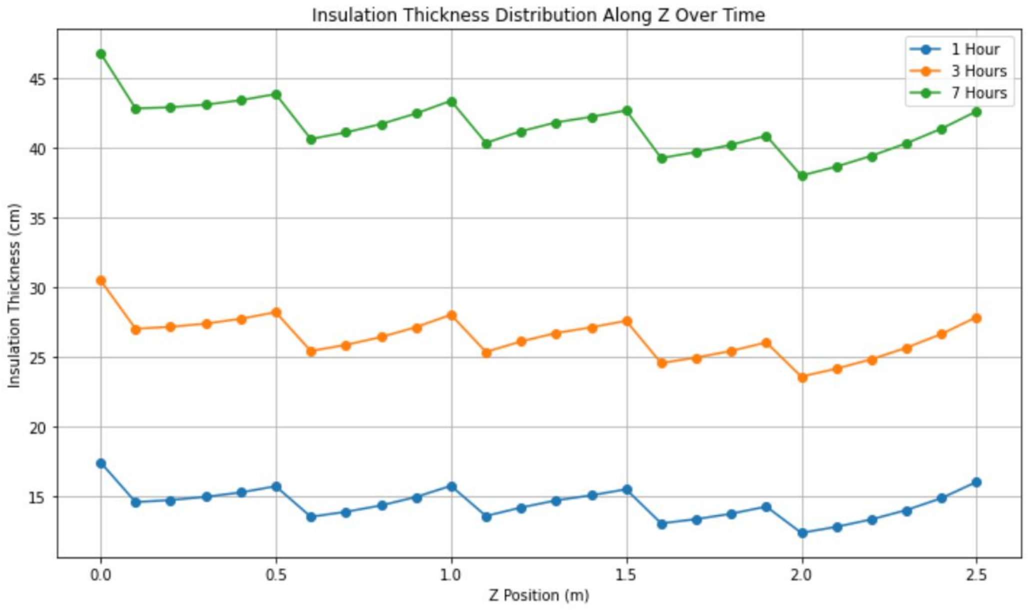
TEXAS A&M UNIVERSITY

- Uses scaled dimensions obj file
- Calculates area at each face
  - Splits quadrilaterals into 2 triangles
- Computes mass at each face
  - m = Area (m^2) x thickness (m) x density (kg/m^3)
- Add up total mass across all faces
  - 1 hr: 49.3702 kg
  - 3 hrs: 90.796kg
  - 7 hrs: 143.202 kg

```
// Handle triangle face.
if (face.indices.size() == 3) {
  const Vertex& v1 = vertices[face.indices[0]];
  const Vertex& v2 = vertices[face.indices[1]];
  const Vertex& v3 = vertices[face.indices[2]];

  face_area = TriangleArea2D(v1, v2, v3);
  avg_thickness = (v1.z + v2.z + v3.z) / 3.0f;
}

// Handle quadrilateral face (split into 2 triangles).
else if (face.indices.size() == 4) {
  const Vertex& v1 = vertices[face.indices[0]];
  const Vertex& v2 = vertices[face.indices[1]];
  const Vertex& v3 = vertices[face.indices[2]];
  const Vertex& v4 = vertices[face.indices[3]];

  float area1 = TriangleArea2D(v1, v2, v3);
  float area2 = TriangleArea2D(v1, v3, v4);
  face_area = area1 + area2;
  avg_thickness = (v1.z + v2.z + v3.z + v4.z) / 4.0f;
}

// Calculate volume and mass.
float volume = face_area * avg_thickness;
float mass = volume * density;
total_mass += mass;
```

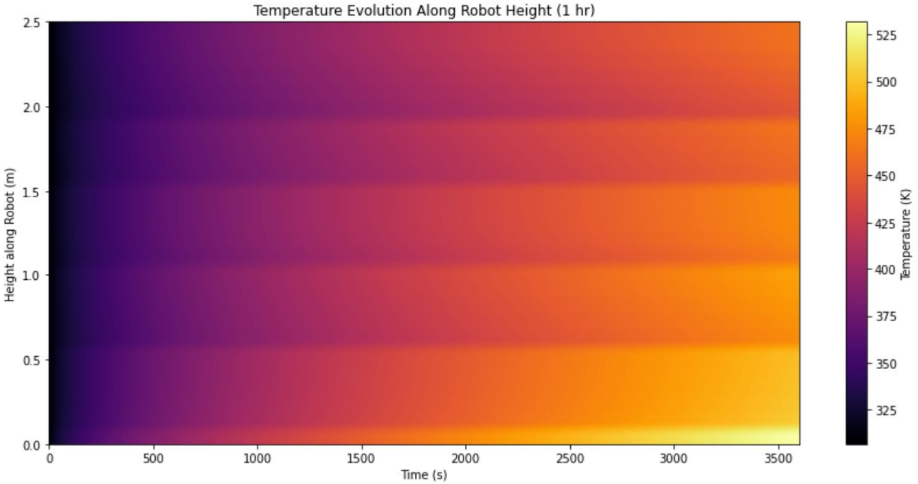
Insulation Thickness Profile Plot

- Thickening varies along z-position, following a periodic pattern.
- Longer submersion time results in greater thickness.
- Average thicknesses for each time interval:
  - 1 hr: 14.43 cm
  - 3 hr: 26.43 cm
  - 7 hr: 41.59 cm

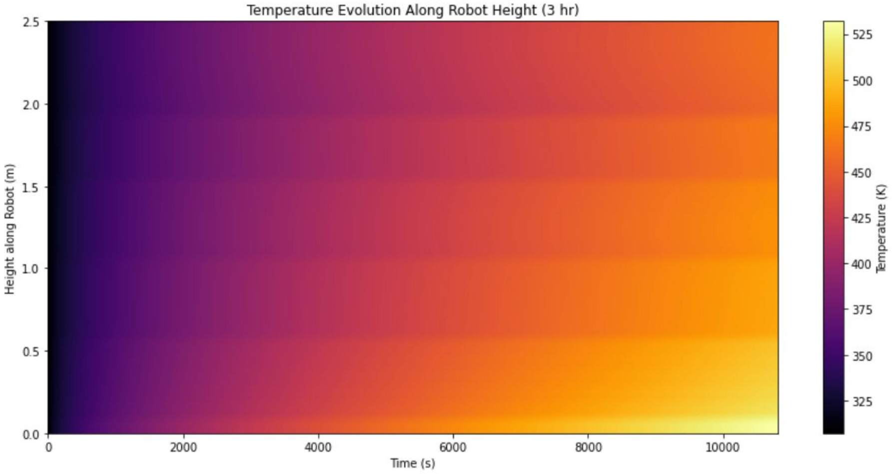


Temperature Evolution Along Robot Height

1 hr:

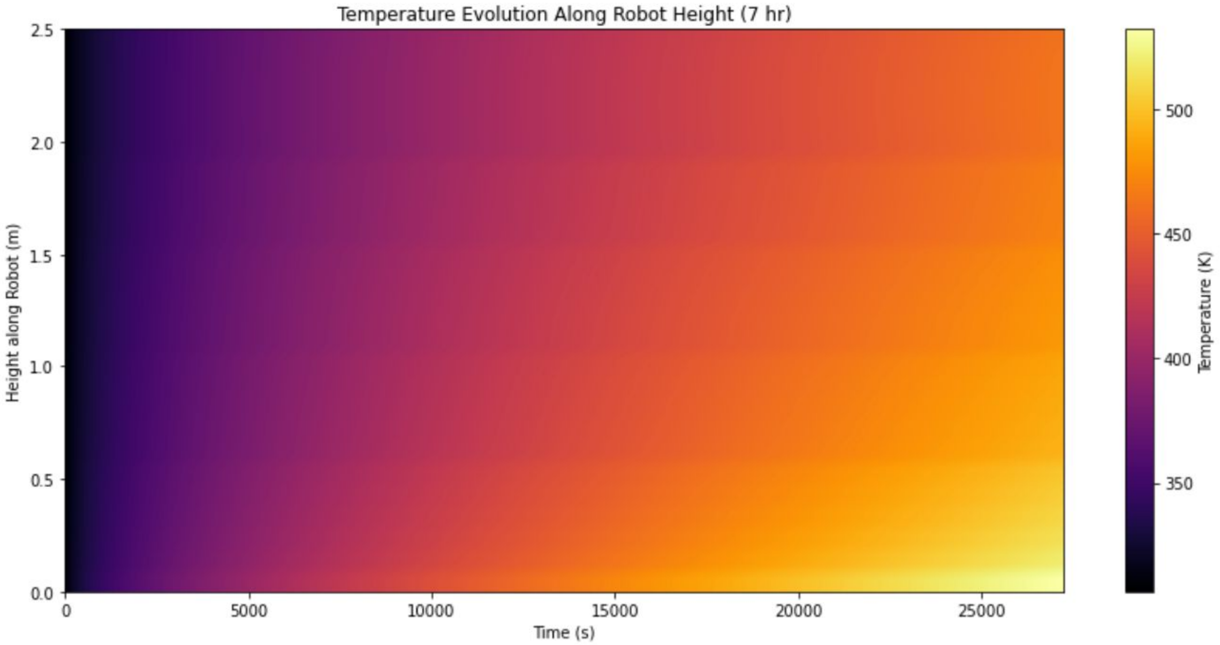


3 hr:

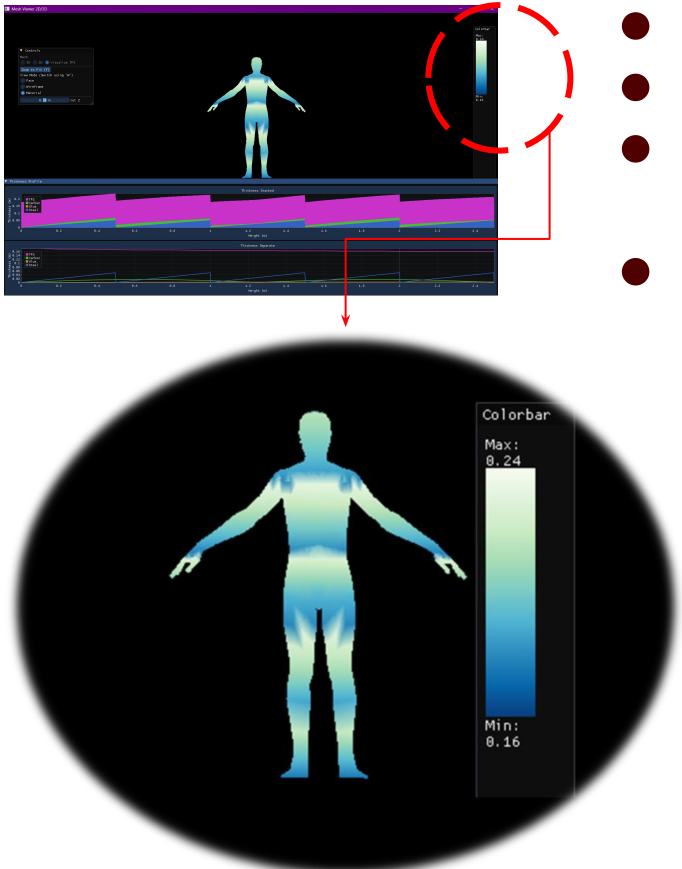


Temperature Evolution Along Robot Height

7 hr:



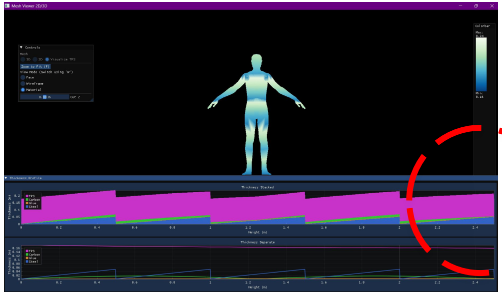
Visualization-Thickness



- Library: **FreeGlut** to **GLFW**
- Mesh modes:**2D**, **3D**, **TPS**.
- View modes: **Face**, **Wireframe**, **Material**.
- **Material** mode visualizes the total thickness with colormap.

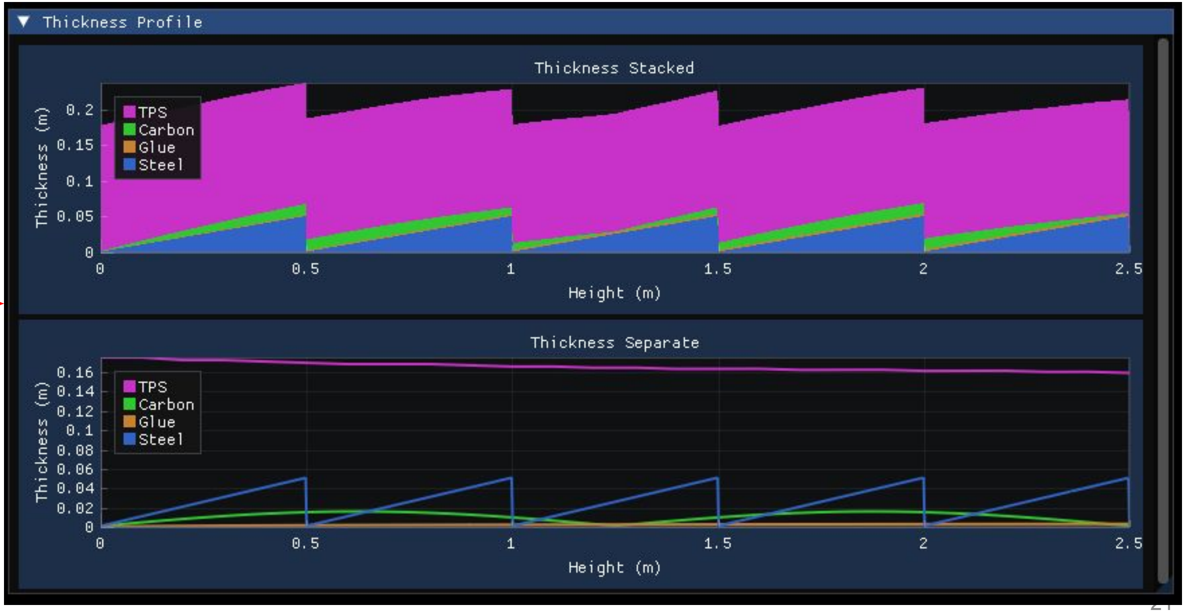
```
for (int i = 0; i < V.size(); i++) {
    const auto& v = V[i];
    float normalized_y_at_v = (v.y - minY) / (maxY - minY);
    float tps_at_v = interp1D(tps_metric_stacked, normalized_y_at_v);

    float normalized_tps_at_v = Normalize(tps_at_v, tps_metric_stacked);
    vertex_colors[i] = InterpolateColormapToVec3(cmap, normalized_tps_at_v);
}

glBindBuffer(GL_ARRAY_BUFFER, CBO);
glBufferSubData(GL_ARRAY_BUFFER, 0, vertex_colors.size() * sizeof(glm::vec3),
```

Visualization-Thickness

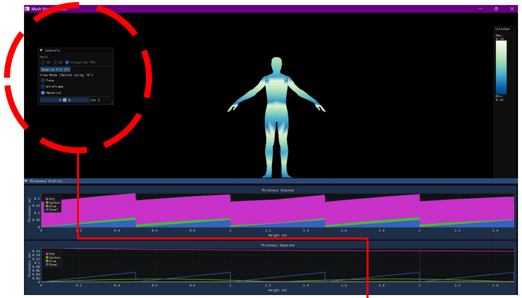


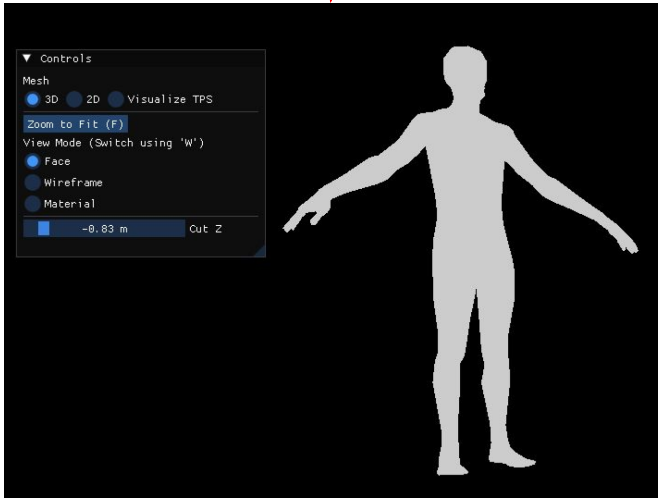
```
ImGui::Begin("Thickness Profile");
if (ImPlot::BeginPlot("Thickness Stacked", ImVec2(-1, plc


    ImPlot::SetupAxisLimits(ImAxis_Y1, 0.0f, *std::max_el
    ImPlot::SetupAxisLimits(ImAxis_X1, robotLength, 0.0f,
    ImPlot::SetupAxis(ImAxis_Y1, "Thickness (m)");
    // ImPlot::SetupAxis(ImAxis_X1, "Height (m)", ImPlotA
    ImPlot::SetupAxis(ImAxis_X1, "Height (m)");

    ImPlot::PushStyleColor(ImPlotCol_Fill, magenta);
    ImPlot::PlotShaded("TPS", heights.data(), tps_metric_
    ImPlot::PopStyleColor();

    ImPlot::PushStyleColor(ImPlotCol_Fill, green);
    ImPlot::PlotShaded("Carbon", heights.data(), carbon_m
    ImPlot::PopStyleColor();

    ImPlot::PushStyleColor(ImPlotCol_Fill, orange);
    ImPlot::PlotShaded("Glue", heights.data(), glue_metri
    ImPlot::PopStyleColor();

    ImPlot::PushStyleColor(ImPlotCol_Fill, blue);
    ImPlot::PlotShaded("Steel", heights.data(), steel_met
    ImPlot::PopStyleColor();
```

- Material thickness is plotted both separately and stacked.



Visualization-GUI





Menu for interaction with GUI.

F key zooms the model to fit.

W key toggles between view modes.

Mouse pan, rotate, zoom supported.

Slider for the cutting plane.

```
ImGui::SetNextWindowSize(ImVec2(280, 0), ImGuiCond_Once);
ImGui::Begin("Controls");

// --- Mesh selection ---------------------------------------

ImGui::BeginDisabled(currentViewMode == MODE_MATERIAL);
ImGui::Text("Mesh");
if (ImGui::RadioButton("3D", currentMeshType == MESH_3D)) {
    loadMeshToGPU(OBJ_3D);
    currentMeshType = MESH_3D;
}
ImGui::SameLine();
if (ImGui::RadioButton("2D", currentMeshType == MESH_2D)) {
    loadMeshToGPU(OBJ_2D);
    currentMeshType = MESH_2D;
}
ImGui::SameLine();
```

Visualization-Error Handling

```
std::vector<float> loadProfile1(const std::string& fname) {
    std::vector<float> vals;
    std::ifstream in(fname);
    if (!in) {
        std::cerr << "[CSV] ERROR: Cannot open file " << fname << "\n";
        return vals;
    }

    std::string line;
    if (!std::getline(in, line)) {
        std::cerr << "[CSV] ERROR: File " << fname << " is empty or inva
        return vals;
    }
```

```
// Check material vectors
if (carbon.empty() || glue.empty() || steel.empty()) {
    std::cerr << "[CSV] ERROR: One or more thickness CSV files are empty.\n";
    return 1; // Exit or handle the error
}
if (carbon.size() != glue.size() || glue.size() != steel.size()) {
    std::cerr << "[CSV] ERROR: Thickness CSV files have inconsistent sizes.\n";
    return 1; // Exit or handle the error
}
```

```
    if (!(ss >> z >> comma >> t) || comma != ',') {
        throw std::runtime_error("Invalid CSV format (expected 'z,t').");
    }

    thermZ.push_back(z);
    thermT.push_back(t);
} catch (const std::exception& e) {
    std::cerr << "[CSV] WARNING: Skipping invalid line: " << line << " (" << e.wha
    continue;
}
```

Hard stop if the file cannot be opened.

Soft skip for individual malformed lines viewer continues with the rest.

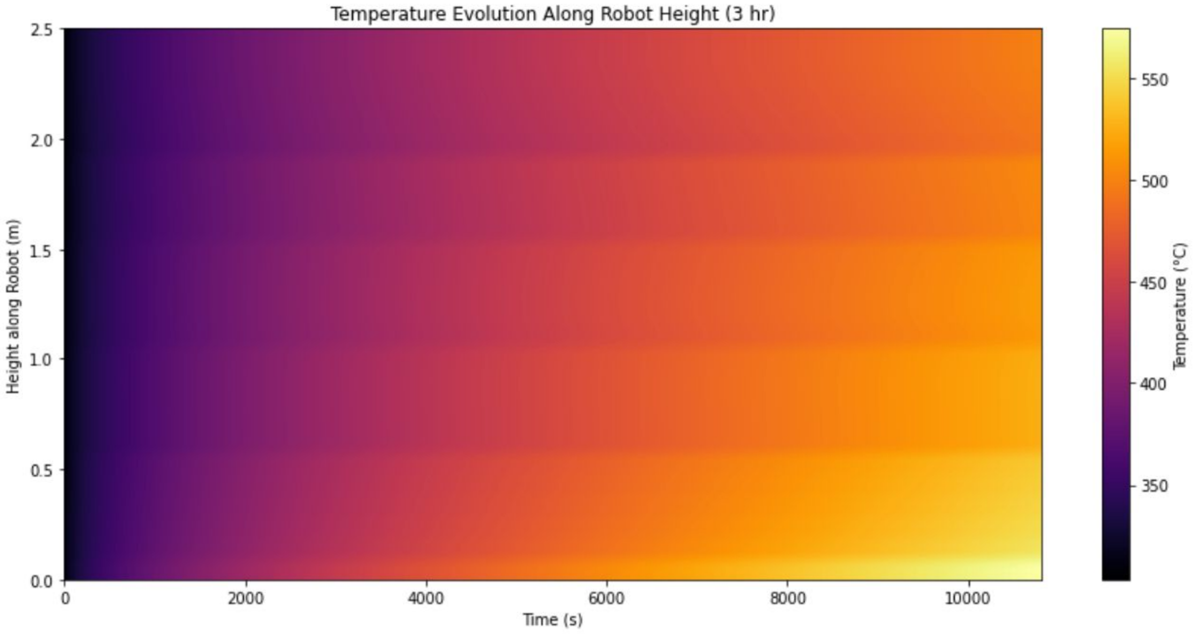
Mesh upload path filters invalid face indices.

Similar pattern is used for CSV and OBJ loading and empty-vector size checks.

Final Simulation Exercise - Results

Total mass of insulation for 3 hours: 72.8 kg

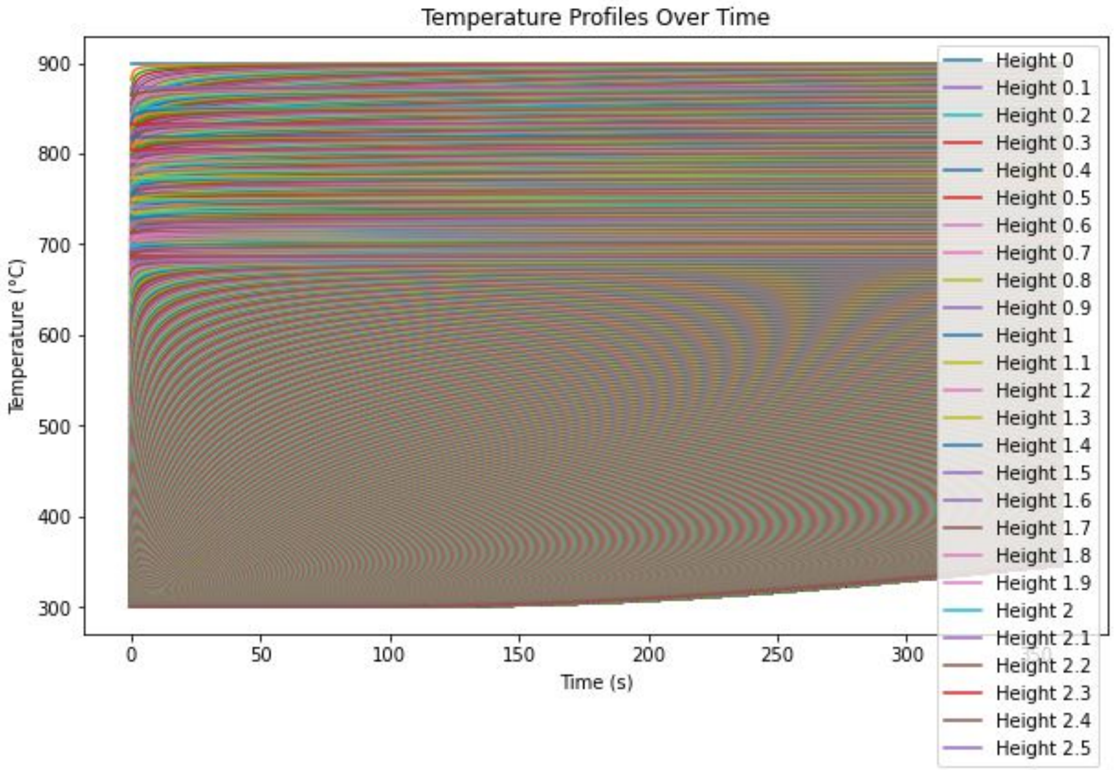
Average thickness of insulation for 3 hours: 21.3015 cm



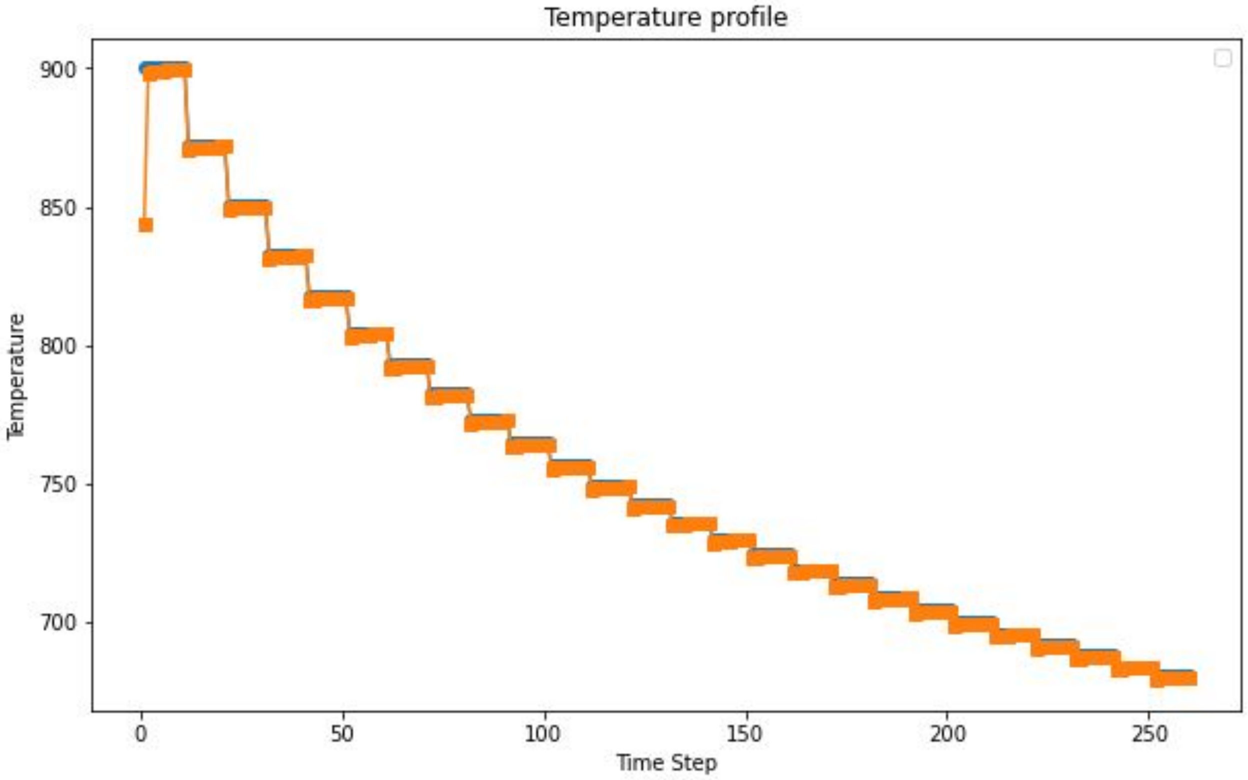
Final Simulation Exercise - Results

Temperature Profile Plot for 1 Hr

- Each z-value block reaches steady state at different times.



Temperature profile at 0 m z value



Authors

TEXAS A&M UNIVERSITY

Member	Contributions
Nishi Mishra	Matrix math for solving heat equation
Layan Samandar	OBJ scaling, thickness parsing, and thermal insulation mass calculation, insulation thickness plot
Parina Patel	OBJ scaling, thickness parsing, and thermal insulation mass calculation, insulation thickness plot
Hassan Niaz	1D heat equation solution - Algorithm selection, implementation and everything in between
Bihao Zhang	Visualization - GUI, thickness plots, color Map CMakeLists, Error Handling
Naveen Jagadeesan	Visualization - GUI Interactive Viewer, Rendering the thickness plots in the viewer, Color Map, Error Handling

Thank You!

Any Questions?