Augmenting Nested Cages with IPC

RIKIN GURDITTA, University of Toronto, Canada
ALEC JACOBSON, Adobe & University of Toronto, Canada

Fig. 1. The funny A guy from the original paper.

Nested cages are multiresolution hierarchies where each mesh in the hierarchy fully encloses the finer next mesh. Existing methods allow for cages to be created with control over application-specific features of the final mesh such as choice of decimation algorithm and “tightness" of enclosing coarse meshes. However, existing methods are bottlenecked by their collision resolution steps. We propose using a modified incremental potential contact approach, using barrier potentials but forgoing continuous collision detection, to generate nested cages with a fast contact step. We use automatic differentiation to simplify the implementation. While we achieve some success with generating cages, our running time is still dominated heavily by contact resolution.

CCS Concepts: • **Computing methodologies** → **Computer graphics**; *Collision detection*; *Mesh geometry models*.

ACM Reference Format:
Rikin Gurditta and Alec Jacobson. 2022. Augmenting Nested Cages with IPC. In *CSC494H1: Computer Science Project, Fall 2022 semester, Toronto, ON*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Multiresolution hierarchies are sequences of shapes that go from high to low resolution. These are important to many tasks in geometry processing, animation, physical simulation, applied mathematics and engineering, and more. Of particular interest are hierarchies of meshes that nest, i.e. where lower resolution meshes entirely contain higher ones beneath them. These often enable computations on sparser data whose results might be efficiently transferred onto more detailed data.

For example, an animator posing a character would like to deform its general pose, and may not be concerned with the fine details of the character’s mesh nor want to wait for computations on a large mesh. Instead, the animator might prefer to deform a lower resolution version of the mesh, and apply their poses to the final mesh in production. Similar ideas can be applied to physics-based animation and PDE solving. Furthermore, completely enclosing coarse meshes are useful for quickly checking for and filtering collisions between meshes - most video games use a special case of this, primitive-shaped bounding volumes.

In this paper we consider how to efficiently and robustly generate 2-dimensional nested cages.

2 RELATED WORK

2.1 Nested Cages

Our method extends the work of Sacht et al. [2015], through implementing a modified version of the method of Li et al. [2020]. The original method treated contact resolution as a “black box" part of the algorithm, while we investigate how IPC may be implemented to best fit the specific case of generating nested cages.

In Nested Cages, there is consideration of the various application-specific energies that can be minimized while constructing the cages, denoted by E_{cages} in our work. Our investigation is instead limited to area energy as we are more interested in the results of a modified contact resolution step, however other application-specific energies are not incompatible with our implementation.

2.2 Simple Nested Cages

Sellán et al. [2022] presents a “simple nested cages" algorithm which takes a totally different approach from Sacht et al. [2015]. The simple nested cages algorithm uses a level set approach to generate tetrahedralizable cages. This algorithm is faster and more robust than Nested Cages, however it achieves this by forgoing user control. Decimations of the fine mesh are not input by the user, instead coarser meshes are decimations of level sets of the signed distance field (SDF) of the fine mesh. No user-defined energy of the coarse mesh is minimized, tightness is always defined by volume.

2.3 Incremental Potential Contact

Our method implements a modified version of IPC [Li et al. 2020], focusing on the use of barrier potentials to handle contact. We forgo implementation of friction altogether since it is not necessary. We also stray from IPC as we do not implement continuous collision detection, our constraint set collection is done by brute force, and we use gradient descent for our solver rather than IPC’s projected Newton method.

CSC494, Dec 26, 2022, Toronto, ON

3 METHOD

In our work we only consider 2D meshes - each mesh $M_k = (\mathbf{V}_k, \mathbf{E}_k)$ consists of a list of vertices $\mathbf{V}_k \in \mathbb{R}^{n_k \times 2}$ and a list of edges $\mathbf{E}_k \in \{1, ..., n_k\}^{m_k \times 2}$. Mathematically, such a mesh describes the boundary of a piecewise-linear bounded shape in $\mathbb{R}^2$. Furthermore, in order to correctly calculate mesh area, we require that mesh edges be correctly oriented. More precisely, considering an edge $e_k = (i, j)$ between vertices $\mathbf{v}_i = (x_i, y_i)$ and $\mathbf{v}_j = (x_j, y_j)$, the normal $\mathbf{n}_k = (y_j - y_i, x_j - x_i)$ should point toward the exterior of the mesh.

The input to the Nested Cages algorithm [Sacht et al. 2015] is a sequence of meshes, $M_0, ..., M_K$ typically going from finest to coarsest, which may overlap. The output is a sequence of meshes $M'_0, ..., M'_K$ which nest, where $M_0 = M'_0$ (this is the finest "original" mesh) and each M_k and M'_k have the same topology, i.e. $\mathbf{E}_k = \mathbf{E}'_k$.

This is done incrementally, by taking consecutive pairs of meshes and making them nest. We start with M'_0 (= M_0) and M_1 and generate M'_1 which completely surrounds M'_0, and then repeat with M'_1 and M_2, etc.

Narrowing our focus to one iteration, consider a fine mesh F (= M'_k) and a coarse mesh C (= M_{k+1}) which may overlap. The algorithm has two main parts: flow and reinflation. As our method follows the same overarching algorithm as Sacht et al's original method, we will focus on the implementation and changes proposed in our method rather than thoroughly presenting the original method.

3.1 Flow

In the flow step, the goal is to shrink the fine mesh F, deforming it until it is entirely within the interior of C. This is done by flowing F along the signed distance field of C. We record and return each flow step we took along the way, i.e. we return a list $[\mathbf{V}_F = \mathbf{V}_F^{(0)}, \mathbf{V}_F^{(1)}, ..., \mathbf{V}_F^{(J)} = \mathbf{V}_F^*]$.

Algorithm 1: Flow

Let $C, F = (\mathbf{V}_F = \mathbf{V}_F^{(0)}, \mathbf{E}_F)$ be the coarse and fine meshes,
h be the step size,
$J_{\max}$ be the max number of iterations before giving up.
$flow\_steps = [\mathbf{V}_F]$
$i \leftarrow 1$
while *F is not inside C and $i <= J_{max}$* **do**
  $\mathbf{g} \leftarrow$ quadrature($\nabla SDF(C)$ at each $\mathbf{v} \in \mathbf{V}_F$)
  $\mathbf{V}_F^{(i)} \leftarrow \mathbf{V}_F^{(i-1)} - h \cdot \mathbf{g}$
  $flow\_steps \leftarrow$ append($\mathbf{V}_F^{(i)}$ to $flow\_steps$)
  $i \leftarrow i + 1$
if $i = I$ **then**
  throw error
return $flow\_steps$

The gradient of the signed distance field of C at any point $\mathbf{v}$ is the direction from $\mathbf{v}$ to the closest point on C to $\mathbf{v}$. We find this closest point by finding the minimum point-edge distance across all edges in $\mathbf{E}_C$.

This direction may not be well defined or differentiable - there may be multiple closest points, and the point chosen may vary wildly with small perturbations of $\mathbf{v}$. Furthermore, we cannot only query $\nabla SDF(C)$ at $\mathbf{v} \in \mathbf{V}_F$, since this would not take into account the directions in which the other points along the edges incident to $\mathbf{v}$ must move. These problems are all solved by numerical quadrature.

For each $\mathbf{v} \in \mathbf{V}_F$, we perform quadrature by calculating a weighted sum of $\nabla SDF(C)$ sampled evenly along the two edges incident to $\mathbf{v}$, weighted quadratically by far across the edges they are.

Algorithm 2: Naïve uadrature along incident edges

Let C be the coarse mesh,
$\mathbf{v} \in \mathbf{V}_F$ be the point at which we want to calculate $\nabla SDF(C)$,
$\mathbf{w}_0, \mathbf{w}_1 \in \mathbf{V}_F$ be the endpoints of the two edges incident to $\mathbf{v}$,
q be the number of quadrature points we are using along each edge.
$\mathbf{g}_\mathbf{v} \leftarrow 0$
for *i from 0 to $q - 1$* **do**
  `// calculate progress along edge`
  $t \leftarrow i/q$
  `// calculate point` t`% along edge from` $\mathbf{v}$ `to` $\mathbf{w}_0$
  $\mathbf{p}_0 \leftarrow (1-t)\mathbf{v} + t\mathbf{w}_0$
  `// scale` $\mathbf{p}_0$`'s contribution by distance from` $\mathbf{v}$
  $\mathbf{g}_\mathbf{v} \leftarrow \mathbf{g}_\mathbf{v} + (1-t)^2(\nabla SDF(C)$ at $\mathbf{p}_0)$
  `// same along edge from` $\mathbf{v}$ `to` $\mathbf{w}_1$
  $\mathbf{p}_1 \leftarrow (1-t)\mathbf{v} + (1-t)\mathbf{w}_1$
  $\mathbf{g}_\mathbf{v} \leftarrow \mathbf{g}_\mathbf{v} + (1-t)^2(\nabla SDF(C)$ at $\mathbf{p}_1)$
return normalize($\mathbf{g}_\mathbf{v}$)

To calculate our stopping condition - whether F is entirely contained within C - we check whether all points point $\mathbf{v} \in \mathbf{V}_F$ are inside C by calculating their winding number mod 2. If all winding numbers are odd then $\mathbf{V}_F$ is inside C, but some edges may partially fall outside C, so we check if F and C intersect.

Sacht et al. [2015] suggest simply checking one point and then intersection. However, This does not work if F has multiple connected components, such as a "hole". The point on F whose winding number is queried may be on a connected component that is entirely within C, and there may be another connected component entirely outisde of C. In this situation the test would incorrectly pass.

3.2 Reinflation

In the reinflation step, we reverse the flow - that is, we start with $F^t = (\mathbf{V}_F^t = \mathbf{V}_F^*, \mathbf{E}_F)$, deform its vertices into $\mathbf{V}_F^{(J-1)}$, etc until we've eventually deformed $\mathbf{V}_F^t$ into $\mathbf{V}_F$. We also start with $C^t = C$. Along the way, we calculate collisions between F^t and C^t, and we deform C^t to continue to surround our F^t. Thus we end up with $C' = (\mathbf{V}'_C, \mathbf{E}_C)$, a deformation of C which surrounds F.

This step is implemented as gradient descent to minimize weighted energies. For a time step t while flowing between $\mathbf{V}_F^{(i+1)}$ and $\mathbf{V}_F^{(i)}$, we have a reinflation energy R_i based on how far away $\mathbf{V}_F^t$ is from its target $\mathbf{V}_F^{(i)}$, an application-specific "nested cages energy" E_{cages}

typically based on how tightly $\mathbf{V}_C$ fits around $\mathbf{V}_F^t$, a barrier potential B based on whether the meshes are colliding, and a containment energy E_{cont} based on whether the current fine mesh is completely within C. Some of these energies have corresponding weights. We use $\mathbf{x}$ to represent the state of the system, i.e. the positions of all vertices of $\mathbf{V}_F^t$ and $\mathbf{V}_C^t$.

$$E(\mathbf{x}) = \lambda_R R_i(\mathbf{x}) + \lambda_{\text{cages}} E_{\text{cages}}(\mathbf{x}) + \lambda_B B(\mathbf{x}) + E_{\text{cont}}(\mathbf{x})$$

We take a fixed number of descent steps. For each step, our solver uses backtracking line search [Boyd and Vandenberghe 2004] to decide an appropriate descent step size - it initially chooses a large step size, and exponentially decreases it until the step leads to a valid lower energy state.

3.2.1 Reinflation energy. The reinflation energy is calculated as

$$\mathbb{R}_i(\mathbf{x}) = \frac{\left\| \mathbf{V}_F^{(i)} - \mathbf{V}_F^t \right\|^2}{2}$$

so its gradient points in the direction from $\mathbf{V}_F^t$ to $\mathbf{V}_F^{(i)}$ and its Hessian is the identity matrix (with respect to $\mathbf{V}_F^t$).

3.2.2 Cages energy. In our method, the nested cages energy is the area of the coarse mesh. As demonstatrated by Sacht et al. [2015] with their analogous volume energy, keeping this as small as possible while maintaining that C^t surrounds F^t means that C^t always wraps tightly around F^t. This is typically desirable in multiresolution applications.

Using Green's theorem, we can calculate the area of a 2D shape by computing a line integral over its boundary. Since C^t is a mesh, its boundary is piecewise-linear, simplifying this line integral and allowing us to find a closed-form solution (see Appendix A):

$$\text{Area}(C^t) = \sum_{(i,j)\in\mathbf{E}_C} \frac{(y_j - y_i)(x_i + x_j)}{2}$$

where each oriented edge $(i, j) \in \mathbf{E}_C^t$ connects vertices (x_i, y_i) and $(x_j, y_j) \in \mathbf{V}_C^t$. (Note that orientation is important here, as inconsistent orientation would lead to contributions from some edges being negated, and opposite orientation would result in a negative area.)

This closed form solution as a summation over the edges of the mesh also lends itself well to computation using automatic differentiation tools such as TinyAD [Schmidt et al. 2022].

Since we are using a descent based optimization method with line search, it is possible that a large step would lead to a state where the coarse mesh is "inverted", where its normals point inward or where the boundary of a "hole" surrounds the outer boundary. In this case, the computed area would be negative which would be disastrous to our optimization. Anyway, such a state would be invalid, as we would like to maintain the quality of our coarse mesh. To prevent this, we return infinite energy when we encounter negative areas. So our nested cages energy is

$$E_{\text{cages}}(\mathbf{x}) = \begin{cases} \text{Area}(C^t) & \text{Area}(C^t) \geq 0 \\ \infty & \text{Area}(C^t) < 0 \end{cases}$$

3.2.3 Barrier potential. We use the barrier potential presented by Li et al. [2020] with point-edge primitive pairs. For two meshes A and B, given a maximum distance $\hat{d}$, for each vertex $\mathbf{v}$ of A we find all edges e of B that are at most $\hat{d}$ away from $\mathbf{v}$. Then calculating the distance d between $\mathbf{v}$ and e, we add to our potential $-(d - \hat{d})^2 \log(d/\hat{d})$. This sum over all pairs $(\mathbf{v}, e)$ makes up our barrier potential $B(A, B)$. This potential has a vertical asymptote near $d = 0$, making it blow up as vertices of A get too close to B. Note that this is not symmetric, i.e. $B(A, B) \neq B(B, A)$.

In our case, we want to prevent F^t and C^t from colliding, so we add $B(F^t, C^t)$ and $B(C^t, F^t)$ to our total energy. We also want to prevent C^t from intersecting itself, so we add $B(C^t, C^t)$ to our energy. Abusing notation, we get our final barrier potential term

$$B(\mathbf{x}) = B(F^t, C^t) + B(C^t, F^t) + B(C^t, C^t)$$

We also calculate gradients and Hessians for this term using TinyAD.

3.2.4 Containment energy. During our reinflation, we want F^t to remain completely contained within C^t. Notably, barrier energy is agnostic of which side of the barrier a point is on, so with large steps it is possible for a vertex of F^t to end up outside of C^t, or for a vertex of C^t to end up inside of F^t. Even aside from these cases, as demonstrated in Figure 3.1, containment by checking vertices is fraught. To prevent our optimization from stepping to a state where F^t is not contained in C^t, we add an additional containment energy.

$$E_{\text{cont}}(\mathbf{x}) = \begin{cases} 0 & F^t \text{ is contained in } C^t \\ \infty & \text{otherwise} \end{cases}$$

We calculate it using the same method as the stopping condition for the flow step. We do not calculate gradients or Hessians for this term, as just like our case-based cages energy, this term is meant to help our line search reject invalid states.

This method of avoiding intersections is our substitute for IPC's intersection-aware line search filter [Li et al. 2020], which itself modifies the line search filter presented by Smith and Schaefer [2015].

3.2.5 Weights. There is no weight attached to the containment energy, since it only takes values from $\{0, \infty\}$. The weight λ_B on the barrier potential is in effect another control over the distance that is maintained between F^t and C^t, on top of its own parameter $\hat{d}$.

The weights most critical to our optimization are λ_R and λ_{cages} controlling the influence of the reinflation and cages energies respectively. In our case, where E_{cages} is the area of the mesh, these energies are directly competing as V^t tries to flow outward and expand while C^t tries to shrink inward and stay as small as possible. The numeric values of these energies depend on the scale and density of the input meshes. For example, a fine mesh with more vertices and at greater scale may have to move a greater distance to reverse its flow. Similarly, a coarse mesh at greater scale or with fewer holes will have greater area

4 RESULTS

We evaluate results of one step of the algorithm, since generating a sequence of cages inductively follows. We find our implementation effective for meshes that succeed in the flow step.

Fig. 2. The fine mesh is shown in green and the coarse mesh is shown in red. Their original positions are on the left, the result of the flow step is in the centre, and the reinflated fine mesh with nested cage is on the right.

Fig. 3. Another example of a successful nested cage.

Fig. 4. As required, nested cages can be generated when the fundamental group of the coarse mesh is a subgroup of the fundamental group of the fine mesh.

We measure performance on a 2022 MacBook Air with an M2 processor and 8 GB of RAM. Test our disc example, using a constant size coarse mesh (with 32 vertices) and increasing the number of vertices of the fine mesh.

Table 1. Variation in performance seems to follow a clearer trend as fine meshes have more vertices.

$\#V_F$	flow time (s)	reinflation time (s)
32	0.000632	4.24838
48	0.000442	2.84045
64	0.00094	6.51258
96	0.000747	4.08192
128	0.001047	5.13875
192	0.00133	5.95799
256	0.00188	7.8915
512	0.002275	10.2327

We can see in Table 1 that the reinflation step still dominates the running time of the program to an extreme extent.

5 LIMITATIONS AND FURTHER WORK

The main limitation of our work is a lack of critical comparison of the performance and robustness differences between our method and the original method, which used the El Topo library [Brochu and Bridson 2009] for its contact resolution. We make suggestions below for further work on optimization of our method that may result in speedups for the flow and reinflation steps in tandem; it is interesting to investigate how Nested Cages and IPC may be melded together but without a baseline comparison it is impossible to tell if this is worthwhile.

The first technical limitation of our work stems from its scope: We are focused on improving the reinflation step of the algorithm. However, Sacht et al. [2015] describe how the flow step is not entirely robust, as SDF flow cannot handle any pair of coarse and fine meshes. Our method suffers from the same pitfall as our flow step is fundamentally identical. Particularly, we can observe failures for extremely coarse C (see Figure 5).

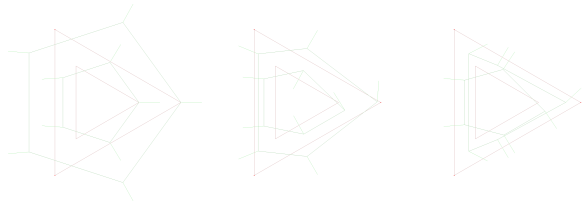

Fig. 5. Just as in Sacht et al. [2015], the flow step may fail for very coarse meshes. In this case, the inner boundary of the fine mesh gets stuck around the inner boundary of the fine mesh.

The quadrature scheme used in our flow step was chosen experimentally, and a better motivated quadrature scheme may lead to faster flow. In Nested Cages [Sacht et al. 2015], the barycentre of the 3D triangle faces is used, this was not adequate for our meshes experimentally. Furthermore, the contributions of points sampled falls off quadratically by their progress along the edge. This weighs sample at the queried vertex much higher than the samples around it. This was done because otherwise, it is possible for most of the incident edges to cross into the coarse mesh but for the vertex itself to remain outside. Also, in both our work and in Nested Cages, an equal number of quadrature points is used per element, which results in lower density of points at parts of the mesh with larger elements. We are only performing quadrature along F, not C, so parts of F with low detail (large elements) may not choose good directions to flow in when they are near parts of C with high detail. These problems could be improved by sampling quadrature points in a more evenly spaced way around the mesh, and also sampling different points each time.

Further work to improve efficiency could start by introducing object partitioning throughout the flow and reinflation steps. In both of these steps we calculate all pairs of vertex-edge distances between meshes: in the flow step (when calculating $\nabla SDF(C)$) we look for the minimum distance, and in the reinflation step (when calculating $B_t(F^t, C^t)$, $B_t(C^t, F^t)$, $B_t(C^t, C^t)$) we look for pairs that have distance less than a threshold $\hat{d}$. Both of these could be sped up by avoiding distance computation for pairs that are definitely too far apart. IPC accomplishes this through “a combined spatial hash and distance filtering structure" [Li et al. 2020].

Furthermore, a data structure such as a bounding volume hierarchy (e.g. an AABB tree) would lend itself well to reuse. A BVH would be calculated for each $F^{(j)}$ in the flow part of the algorithm, and could be reused for in the reinflation step when $F^t = F^{(j)}$ as it reverses the flow. In the reinflation step, the bounding volume hierarchies for F^t and C^t could each be calculated once per timestep and reused for the three barrier potential terms.

Our method also diverges from IPC as we do not use continuous collision detection. IPC uses a line search filter inspired by the one proposed by [Smith and Schaefer 2015]. It caches the minimum distance between meshes while calculating the barrier energy, and uses this distance as an upper bound on step size. This reduces the number of line search iterations required. This would help the running time of our method as it would decrease the number of expensive total energy computations we would perform.

Lastly, our method diverges from IPC most critically by using gradient descent for its optimization rather than IPC's barrier-aware projected Newton solver.

This gradient descent is itself not optimized well. By taking a fixed number of descent steps, we may waste time stalling, i.e. evaluating and taking steps when we are already at our target. This fixed number also needs to be tuned by hand per mesh, making the program less automatic. We suggest fixing this by descending and progressing V_F^t through the flow until it reaches its target $V_F^{(i)}$, or until it hits a predetermined large max number of iterations indicating that it is unable to make reasonable progress.

ACKNOWLEDGMENTS

To my cat Shadow, who yelled at me and sat on my computer all semester as I worked on this.

REFERENCES

Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press, The Edinburgh Building, Cambridge, CB2 8RU, UK. https://web.stanford.edu/~boyd/cvxbook/

Tyson Brochu and Robert Bridson. 2009. Robust Topological Operations for Dynamic Explicit Surfaces. *SIAM Journal on Scientific Computing* 31, 4 (2009), 2472–2493. https://doi.org/10.1137/080737617 arXiv:https://doi.org/10.1137/080737617

Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. 2020. Incremental Potential Contact: Intersection- and Inversion-free Large Deformation Dynamics. *ACM Trans. Graph. (SIGGRAPH)* 39, 4, Article 49 (2020), 20 pages.

Leonardo Sacht, Etienne Vouga, and Alec Jacobson. 2015. Nested Cages. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 14 pages.

Patrick Schmidt, Janis Born, David Bommes, Marcel Campen, and Leif Kobbelt. 2022. TinyAD: Automatic Differentiation in Geometry Processing Made Simple. *Computer Graphics Forum* 41, 5 (2022), 12 pages.

Silvia Sellán, Jack Luong, Leticia Mattos Da Silva, Aravind Ramakrishnan, Yuchuan Yang, and Alec Jacobson. 2022. Breaking Good: Fracture Modes for Realtime Destruction. *ACM Transactions on Graphics* (2022), 12 pages.

Jason Smith and Scott Schaefer. 2015. Bijective Parameterization with Free Boundaries. *ACM Trans. Graph.* 34, 4, Article 70 (2015), 9 pages. https://doi.org/10.1145/2766947

A AREA FORMULA FOR A SHAPE WITH PIECEWISE LINEAR BOUNDARY

Suppose C is a shape with piece-wise linear boundary $\mathbf{E} = (e_1, ..., e_m)$, each edge $e_k = (i, j)$ representing points (x_i, y_i) and (x_j, y_j). Then we can calculate its area using Green's theorem

$$\begin{aligned}\text{Area}(C) &= \iint_C 1 dA \\ &= \int_{\partial C} x dy \\ &= \int_{\partial C} \begin{pmatrix} 0 \\ x \end{pmatrix} \cdot \begin{pmatrix} dx \\ dy \end{pmatrix}\end{aligned}$$

Breaking this up between pieces of the boundary,

$$\begin{aligned}&= \sum_{(i,j)\in\mathbf{E}} \int_0^1 (x_i + t(x_j - x_i))(y_j - y_i) dt \\ &= \sum_{(i,j)\in\mathbf{E}} (y_j - y_i) \left[x_i t + \frac{x_j - x_i}{2} t^2 \right]_0^1 \\ &= \sum_{(i,j)\in\mathbf{E}} \frac{(y_j - y_i)(x_i + x_j)}{2}\end{aligned}$$

Received 26 December 2022