A high order stable conservative method for solving hyperbolic conservation laws on arbitrarily distributed point clouds

Jie Du[1] and Chi-Wang Shu[2]

Abstract

In this paper, we aim to solve one and two dimensional hyperbolic conservation laws on arbitrarily distributed point clouds. The initial condition is given on such a point cloud, and the algorithm solves for point values of the solution at later time also on this point cloud. By using the Voronoi technique and by introducing a grouping algorithm, we divide the computational domain into non-overlapping cells. Each cell is a polygon and contains a minimum number of the given points to ensure accuracy. We carefully select points in each cell during the grouping procedure, and hence are able to interpolate or fit the discrete initial values with piecewise polynomials. By adapting the traditional discontinuous Galerkin method on the constructed polygonal mesh, we obtain a stable, conservative and high order method. Numerical results for both one and two dimensional scalar equations and Euler systems of compressible gas dynamics are provided to illustrate the good behavior of our mesh generation algorithm as well as the numerical scheme.

Key Words: arbitrarily distributed point cloud, conservation laws, high order, discontinuous Galerkin.

[1]Department of Mathematics, Chinese University of Hong Kong, Hong Kong, P.R. China. E-mail: jdu@math.cuhk.edu.hk

[2]Division of Applied Mathematics, Brown University, Providence, RI 02912, USA. E-mail: shu@dam.brown.edu. Research supported by AFOSR grant F49550-12-1-0399 and NSF grant DMS-1418750.

1 Introduction

In this paper, we are interested in solving the following hyperbolic conservation law

$$u_t + f(u)_x = 0, \tag{1}$$

with suitable initial condition $u(x, 0)$, and its two-dimensional version

$$u_t + f(u)_x + g(u)_y = 0 \tag{2}$$

with suitable initial condition $u(x, y, 0)$. Here, u, f and g can be either scalars or vectors. Unlike traditional finite difference (FD) methods for which a structured set of grid points is given, or traditional finite volume (FV), finite element (FE) or discontinuous Galerkin (DG) methods for which a structured or unstructured mesh is given, we assume that an arbitrarily distributed point cloud (a set of unstructured points), together with the values of the initial condition at these points, is given, and we seek an algorithm to obtain the point values of the numerical solution in this point cloud for later time. One possible scenario for such a set-up could be that the point clouds are locations of the observation posts or places where measurements are being made, and evolution data would need to be predicted and compared with future measurements.

This problem is not directly suited to classical well developed computational methods such as FD methods, FV volume methods and FE methods. The underlying structure of these methods is that they first cover the computational domain with a given grid or mesh and then build the algorithm upon it. For example, in the FD method, by knowing the connectivity between neighbors in the grid, we can use values on some points of the grid to compute difference operators to approximate the derivatives and hence the partial differential equations (PDEs). Another difficulty is that, unlike traditional problems where the initial condition is assumed given as a function, here we only assume the knowledge of the initial values on the arbitrarily distributed point cloud. Our algorithm seeks the numerical solution on the same point cloud for later time. Hence it appears difficult to apply the classical grid- or mesh-based numerical methods directly.

Meshless methods are alternatives to traditional mesh-based methods, which may be suitable for our problem. The objective of these methods is to provide numerical solutions in terms of nodes without using any mesh to connect them or using a background mesh only minimally, for example, only for the numerical integration. There are many different types of meshless methods, such as the smoothed particle hydrodynamics (SPH) [34, 6, 22, 36], the diffuse element method (DEM) [8], the reproducing kernel particle method (RKPM) [33] and the partition of unity method [35]. An important component in the meshless method is the meshless interpolation. It approximates functions based on a set of scattered points that have no particular topological connection among them, such as the Shepard's interpolant [42] and the moving least square method [31, 37]. A significant progress has been made for these methods through the work of Babuška and Melenk [4]. They recognized that the methods based on moving least squares are specific instances of partitions of unity. For a broad survey of the meshless methods, we refer to [32].

Meshless methods have been used in many different applications. However, most of them are mainly concerned with engineering applications, focusing on the practical aspects of the algorithms without extensive analysis to issues like conservation, accuracy and stability. There are a few papers emphasizing the analysis of these methods for solving PDEs, such as [2, 3] for steady-state linear elliptic equations, [25] for steady convection dominated problems, and [43] for incompressible Navier–Stokes equations. However, to our best knowledge, there are few papers devoted to meshless methods for solving time-dependent hyperbolic conservation laws, and conservation and stability appear to be particularly difficult for meshless methods for such PDEs. [48] uses an improved localized radial basis functions collocation method (LRBFCM) for the numerical solution of hyperbolic Burgers equation. However, they have chosen uniform nodal arrangement due their suitability and better accuracy.

We would like to solve time-dependent hyperbolic conservation laws under the same condition as meshless methods, however we would like to still utilize traditional grid-

or mesh-based methods which have many important good properties. Since we would like to solve time dependent conservation laws on arbitrarily distributed point clouds, we would need to find a way to generate suitable grid or mesh and interpolate or fit the discrete initial values with functions in order to use the traditional methods. We start from building an appropriate mesh based on the given point cloud, so that each cell is a polygon, and contains a minimum number of points in the original point cloud so that a polynomial of a pre-defined degree can be constructed to represent the initial condition to high order accuracy. Once the polygonal mesh is constructed, we march the piecewise polynomial numerical solution in time by choosing a suitable numerical method.

Note that the polygonal meshes are difficult for traditional finite element methods as it is difficult to construct continuous finite element spaces on such cells (see, for example, the recent development of virtual element methods [5] which uses finite element spaces with a high number of degrees of freedom per cell and non-polynomial basis functions in order to obtain continuity on polygonal mesh interfaces). Finite volume methods are also difficult because reconstruction on such heterogeneous polygonal meshes (different cells may have different number of sides) is difficult. For discontinuous Galerkin (DG) methods, such difficulties do not exist and we can use the usual piecewise polynomials of degree k for any pre-defined k as our finite element space. Hence, DG is a relatively suitable method on such heterogeneous polygonal meshes. As far as we know, there are few papers discussing DG methods on such heterogeneous polygonal meshes. [17] presents an explicit DG method applied to a nonlinear convection-diffusion equation, which allows the usage of a nonconforming mesh formed by non-convex star-shaped polyhedral elements. As a continuation, [16] proposes a semi-implicit DG scheme and presents numerical experiments to treat the effect of non-convexity of elements and nonconformity of a mesh. Besides the DG method, other methods such as the CPR method [28, 49, 18, 19] could of course also be used on the polygonal meshes, but it appears that the DG method has the best conservation, stability and accuracy properties.

We point out that our method is indeed based on the same setup as meshless methods, that is, the initial data is given on a set of arbitrarily distributed points and we seek for the values on these points for later time. Unlike most meshless methods, our paper presents a way to cover the computational domain with a suitable polygonal mesh and to construct suitable initial functions for DG time marching. Besides the generation of the mesh, we also need to compute cell and edge integrals for polygons, which may be costly. However, due to the good properties of the DG method, our method is conservative, stable and high order accurate, both for linear and nonlinear equations. These properties are difficult to achieve simultaneously by using previous meshless methods. Hence, when these properties are desired, our scheme may be a good choice despite of its relatively higher computational cost.

This paper is organized as follows. In Section 2, we describe how to generate a suitable mesh as well as how to interpolate or fit the given values at the original point cloud to obtain a piecewise polynomial at the initial time. In Section 3, a brief review of the DG scheme is given to illustrate how we march in time. Also, a more detailed comment on the computational cost and the good properties of our method will be given in this section. In Section 4, numerical experiments are provided to verify the accuracy and stability of our method. We also test the discontinuous initial conditions in this section. Finally, concluding remarks are provided in Section 5.

2 Mesh generation

Let us denote the computational domain as Ω and assume that the initial condition is given as point values on a point cloud, namely a finite set of distinct, isolated points $\{\mathbf{z}_i\}_{i=1}^N$ belonging to Ω. A set $\{V_j\}$ is called a tessellation of Ω if $V_j \bigcap V_k = \emptyset$ for $j \neq k$ and $\bigcup_j \overline{V}_j = \overline{\Omega}$. With the given set of the random points $\{\mathbf{z}_i\}_{i=1}^N$, we first need to generate a tessellation of Ω before we can actually proceed with our numerical scheme. Our goal is that each cell V_j in the tessellation contains at least a minimum number of

the given points such that we can obtain a good approximation to the exact solution by interpolating or fitting the given values at these points.

The mesh generation includes two steps. The first step is to adopt the Voronoi tessellation technique to divide the computational domain into small regions. Each region contains one single given point. We will review the concept of the Voronoi tessellation [20, 38] in Section 2.1. The second step is to group these small Voronoi regions into cells such that we can interpolate or fit the discrete initial values with piecewise polynomials, thus achieving high order accuracy. The algorithm of grouping Voronoi regions into cells will be illustrated in Section 2.2.

2.1 Voronoi diagram

For each point in the point cloud $\{\mathbf{z}_i\}_{i=1}^N$, there is a corresponding region consisting of all locations in Ω closer to that point than to any other point in the cloud with respect to the Euclidean distance. We call the set of locations assigned to each point as its Voronoi region. By denoting $|\cdot|$ as the Euclidean norm, we can write the Voronoi region $\widehat{V}_i$ corresponding to the point $\mathbf{z_i}$ more precisely in mathematical terms as

$$\widehat{V}_i = \{\mathbf{x} \in \Omega \quad | \quad |\mathbf{x} - \mathbf{z}_i| < |\mathbf{x} - \mathbf{z}_j| \quad for\ j = 1, \cdots, N, j \neq i\}. \tag{3}$$

In this definition, each Voronoi region is an open set. Note that Voronoi regions are all polygons. For a comprehensive treatment, see [38].

One can easily see that every location in Ω belongs to at last one closure of the Voronoi regions, that is, $\bigcup_{i=1}^N \overline{\widehat{V}}_i = \overline{\Omega}$. On the other hand, since we use "$<$" in the definition of the Voronoi regions, we have $\widehat{V}_i \bigcap \widehat{V}_j = \emptyset$ for $i \neq j$. Hence, the set $\{\widehat{V}_i\}_{i=1}^N$ is a tessellation of Ω. We call this tessellation a Voronoi tessellation or a Voronoi diagram. The points $\{\mathbf{z}_i\}_{i=1}^N$ are called generators. Figure 1 shows a Voronoi tessellation corresponding to 16 randomly generated points in a square. Here, we use dots to represent the Voronoi generators.

The same idea is straightforward and simpler in the one-dimensional case. Given a

set of random points in one-dimensional computational domain Ω, the Voronoi domain corresponding to each point is a line segment called a Voronoi line. We easily notice that the boundary point between two adjacent Voronoi lines is the midpoint of the generator points of those Voronoi lines. Figure 2 shows a one-dimensional Voronoi tessellation corresponding to 5 randomly generated points in $[0, 1]$.

There are several ways to generate a Voronoi diagram from a set of generators, such as the Fortune's algorithm and the Lloyd's algorithm. In this paper, we adopt the Fortune's algorithm, which is a sweep line algorithm [21]. Our code is based on the Voronoi diagram generator written in $C++$ by Shane O'Sullivan [39], which is a modified version of Steven Fortune's sweep line algorithm, by fixing a few misprints and memory leak issues of the original C code written by Steven Fortune.

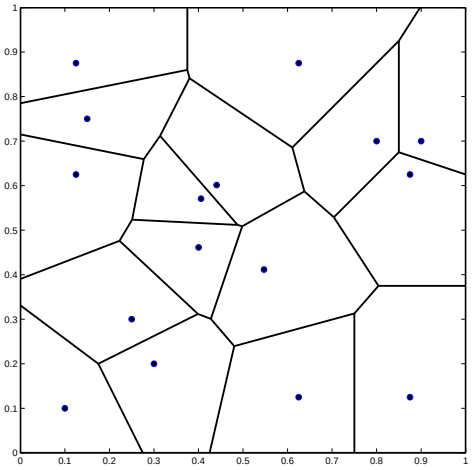


Figure 1: Voronoi diagram in two dimension

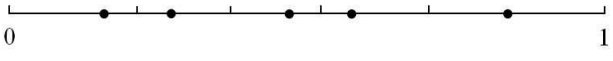


Figure 2: Voronoi diagram in one dimension

2.2 Grouping algorithm

In the last section, we have divided the computational domain Ω into Voronoi regions $\{\widehat{V}_i\}_{i=1}^N$. Each region contains one generator point $\mathbf{z_i}$. To obtain high order accuracy, we now need to further use an appropriate algorithm to group the Voronoi regions into cells. Each cell is a union of several adjacent small Voronoi regions and thus contains the corresponding generator points. All new grouped cells collectively form another tessellation of Ω. We denote them as $\{V_j\}_{j=1}^M$.

In each cell V_j, we would like to interpolate or fit the given discrete initial values by a polynomial in $P^k(V_j)$. Here, $P^k(V_j)$ is the space of polynomials of degree up to k on V_j. For this purpose, the number of generator points in each cell should be at least K, where K is the degree of freedom of $P^k(V_j)$. In the one-dimensional case, $K = k+1$, and in the two-dimensional case, $K = (k+1)(k+2)/2$. Since the total number of generator points $\{\mathbf{z_i}\}_{i=1}^N$ in the entire domain Ω is arbitrary, and because there might be issues of condition numbers for the interpolation polynomials, some cells may contain more than K points. In this case, we use the least squares method to obtain a fitting polynomial.

The locations of the generator points in each cell is crucial. For example, if we need to construct a two-dimensional polynomial in P^1, then the three interpolation points can not be aligned along a straight line. Hence, the choice of the Voronoi regions in each cell can not be arbitrary. Considering the cell V_j, we rename the generator points in this cell as $\{\mathbf{x}_{j,l}\}_{l=1}^L$. By choosing a set of basis functions $\{\phi_m(\mathbf{x})\}_{m=1}^K$ in V_j, we can attempt to interpolate the solution in V_j by

$$u_j(\mathbf{x}) = \sum_{m=1}^{K} \alpha_m \phi_m(\mathbf{x}),\ \mathbf{x} \in V_j, \tag{4}$$

such that

$$u_j(\mathbf{x}_{j,l}) = u_{j,l},\ l = 1, \cdots, L. \tag{5}$$

Here, $u_{j,l}$, $l = 1, \cdots, L$ are the given initial values at the generator points. By denoting $A = (a_{l,m})$ with $a_{l,m} = \phi_m(\mathbf{x}_{j,l})$, $\vec{\alpha} = (\alpha_m)$ and $\vec{b} = (u_{j,l})$, we can rewrite Equations (4)

and (5) into the following matrix version:

$$A\vec{\alpha} = \vec{b}. \tag{6}$$

When the number of generator points in V_j is exactly K, that is, $L = K$, and if A is invertible, we can solve the above system of equations to obtain the interpolation coefficients vector $\vec{\alpha}$. The condition number of A gives a bound on how inaccurate the solution $\vec{\alpha}$ will be. A large condition number will lead to poor solutions. In the extreme situation that the condition number is infinite, Equation (6) is ill-posed, and no algorithm can be expected to reliably find a solution. Hence, a crucial rule in our algorithm of grouping is to bound the condition number of A, by using a threshold value δ.

We can easily see that each row of A relates to one generator point in V_j. During the grouping procedure, even when A is non-square, that is, when the number of points is less than K, we can still compute the condition number of A, which can be viewed as a measure of closeness to a rank loss [15]. If the first several points chosen in V_j have already led to a poor condition number of A, then any choice of the remaining generator points is useless. Hence, we need to make sure that the condition number of A is always bounded by δ each time we add one point into the current cell V_j.

In addition to the condition number issue, another rule is that we would like to make the cell as close to a circle as possible (to have good aspect ratios). Also, we need to make sure that every given point is distributed into one cell such that all cells together form a tessellation of Ω. Considering these issues, we introduce the following algorithm to group Voronoi regions into cells. We show an illustrative figure in Figure 3 with uniformly distributed points to make the algorithm more clear. Here we take $K = 6$ (corresponding to P^2) for example. Note that each generator point in $\{\mathbf{z}_i\}_{i=1}^N$ has an index number i. For the uniformly distributed points, we just sort them in a trivial way as shown in Figure 3. For randomly distributed points, we will compare different indexing of the points in the numerical example in Section 4. It seems that the points

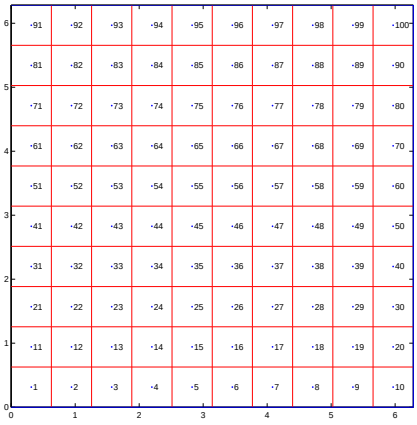
can be sorted arbitrarily.

- We check all generator points one by one from $\mathbf{z}_1$ to $\mathbf{z}_N$. When we check $\mathbf{z}_i$, if it has not been distributed into any cells, we now need to find an appropriate union of $\mathbf{z}_i$ and its neighbors to create a new cell, say, V_j, which should contain K generator points. Let us denote $\mathbf{x}_{j,1} = \mathbf{z}_i$. As shown in Figure 3(b), if we take $i = 1$, then $\mathbf{x}_{1,1} = \mathbf{z}_1$.

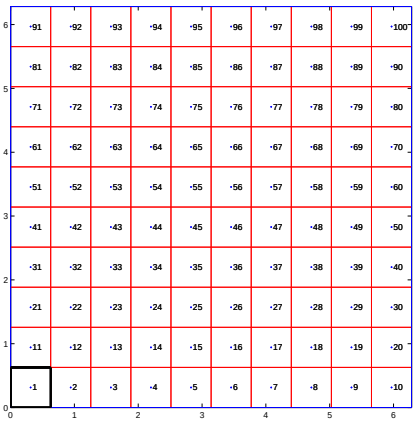
- The next step is to check all immediate neighbors of $\mathbf{x}_{j,1}$ one by one in ascending order of their index numbers. Here and below, two points $\mathbf{x}$ and $\mathbf{y}$ are called immediate neighbors if their Voronoi regions share a common edge. If an immediate neighboring point has not been distributed into any other cells yet and the condition number of the corresponding matrix A is less than δ, then we add this point into the cell V_j. For the example in Figure 3(c), we now have $\mathbf{x}_{1,2} = \mathbf{z}_2$ and $\mathbf{x}_{1,3} = \mathbf{z}_{11}$.

- After we check all immediate neighbors of $\mathbf{x}_{j,1}$, if the number of generator points in V_j is still less than K, we now illustrate how to add remaining points into V_j. We use the following algorithm to add only one point each time, until the total number of points reaches K.

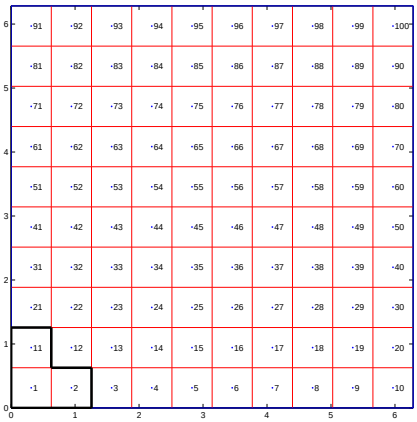
  We find out all available immediate neighboring generator points of V_j (that is, generator points who are immediate neighbors with at least one generator point in V_j), and rank them according to the number of their immediate neighbors in V_j. We are interested in the point with the largest number of immediate neighbors in V_j since it helps to make V_j close to a circle. If there are several points with the same number of immediate neighbors in V_j, we check the one with the smallest index number i first. If the condition number of A related to the existing points in V_j and the above selected point is less than δ, then we add this point into V_j. Otherwise, we check the neighboring point with second largest number of immediate neighbors in V_j, and so on.



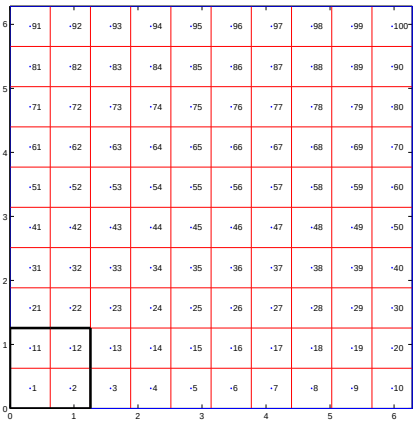
(a)



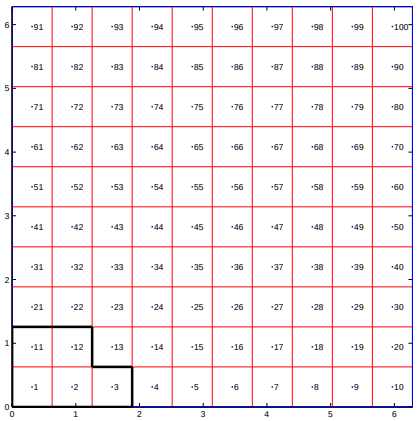
(b)



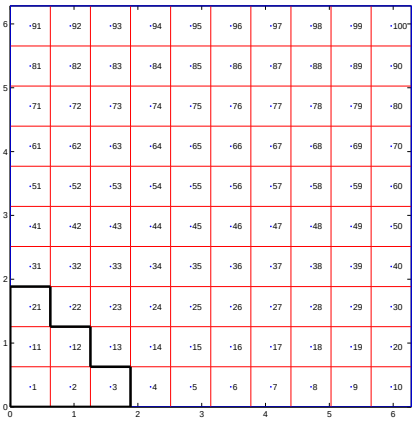
(c)



(d)



(e)



(f)

Figure 3: Procedure to generate V_1

For example in Figure 3, staring from $V_1 = \{\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_{11}\}$, all available immediate neighboring points are $\mathbf{z}_{12}$, $\mathbf{z}_3$ and $\mathbf{z}_{21}$. We check $\mathbf{z}_{12}$ first since it has two immediate neighbors in V_1 while the other two points have only one. The condition number of A related to V_1 and $\mathbf{z}_{12}$ is less than δ. Hence, we add $\mathbf{z}_{12}$ into V_1 and now $V_1 = \{\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_{11}, \mathbf{z}_{12}\}$, as shown in Figure 3(d).

Now, all available immediate neighboring points of V_1 are $\mathbf{z}_3$, $\mathbf{z}_{13}$, $\mathbf{z}_{21}$ and $\mathbf{z}_{22}$. We check $\mathbf{z}_3$ first and it satisfies the condition number limit. Hence, we have $V_1 = \{\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_{11}, \mathbf{z}_{12}, \mathbf{z}_3\}$, as shown in Figure 3(e).

All available immediate neighboring points of the updated V_1 are now $\mathbf{z}_{13}$, $\mathbf{z}_4$, $\mathbf{z}_{21}$ and $\mathbf{z}_{22}$. We check $\mathbf{z}_{13}$ first since it has the largest number of immediate neighbors in V_1. However, it does not satisfy the condition number limit. We move on to check $\mathbf{z}_4$. Again, it fails the condition number test. Next, we check $\mathbf{z}_{21}$ and find that is satisfies the condition number limit. Hence, V_1 becomes $V_1 = \{\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_{11}, \mathbf{z}_{12}, \mathbf{z}_3, \mathbf{z}_{21}\}$, as shown in Figure 3(f). Now the total number is $K = 6$, and hence we can stop the procedure of constructing V_1.

- It is possible that after we have checked all available neighboring points of V_j, the number of points in V_j is still less than K and any choice of additional point will lead to a poor condition number of the corresponding A, that means, we are unable to find K Voronoi regions to compose a good cell starting from the point $\mathbf{x}_{j,1}$, say, $\mathbf{z}_1$. In this case, we put aside $\mathbf{z}_1$ for a while and continue to check $\mathbf{z}_2$. If it has not been grouped into any cell, we set $\mathbf{x}_{j,1} = \mathbf{z}_2$ and use the above algorithm to construct V_j again. After we have checked all generator points, we return to check $\mathbf{z}_1$. If it still has not been added to any cells yet, we add it to the nearest existing cell. By doing so, the number of points in this cell will be larger than K, thus the system of equations (6) is over-determined. In this case, we need to solve the

following least squares problem [7]

$$\min_{\vec{\alpha}} ||\vec{b} - A\vec{\alpha}||_2 \tag{7}$$

to determine the fitting polynomial in (4).

By using the algorithm described in this section, we can divide the computational domain into non-overlapping cells. Each cell contains at least K points in the given point cloud. To achieve a better usage of the given initial values, we have carefully selected the points in each cell during the grouping procedure. Thus, when the number of points is exactly K in a cell, we are able to obtain an interpolation polynomial in P^k by solving Equation (6). When the number of points is more than K, we can also obtain an approximation to the exact solution in P^k by solving the least squares problem (7). In this way, we can recover the initial solution with piecewise polynomials. Note that the algorithm described above is for the two-dimensional case. However, it is trivial to be applied to the one-dimensional case, and hence we omit the details.

3 Numerical method on the generated mesh

In the previous section, we have already divided the computational domain into cells and approximated the initial value with piecewise polynomials. In this section, we need to use a suitable numerical scheme to march in time. We will review the formulation of the Runge-Kutta discontinuous Galerkin (RKDG) method in Section 3.1 and give some comments on our scheme in Section 3.2.

3.1 RKDG discretization

DG methods are a class of finite element methods using completely discontinuous basis functions, which are usually chosen as piecewise polynomials. The first DG method was introduced in 1973 by Reed and Hill [40]. In this section, we will adapt the RKDG method carried out by Cockburn et al. in a series of papers [10, 11, 12, 13, 14]. It uses DG discretization in space and Runge-Kutta method in time.

Recall that we have denoted cells generated in the last section as V_j, $1 \leqslant j \leqslant M$. For the one-dimensional case, we denote $V_j = [x_{j-\frac{1}{2}}, x_{j+\frac{1}{2}}]$ and $\Delta x_j = x_{j+\frac{1}{2}} - x_{j-\frac{1}{2}}$. For the two-dimensional case, V_j are polygons. In the DG method, the numerical solution belongs to the following finite element space consisting of piecewise polynomials

$$V_h^k = \{v : v|_{V_j} \in P^k(V_j), 1 \leqslant j \leqslant M\}. \tag{8}$$

As in the last section, we can interpolate or fit the given initial values to obtain an initial solution in V_h^k.

The DG method in space for the one-dimensional conservation law (1) is defined as: find the unique function $u_h = u_h(t) \in V_h^k$ such that, for all test functions $v_h \in V_h^k$ and all $1 \leqslant j \leqslant M$, we have

$$\int_{V_j} (u_h)_t v_h dx - \int_{V_j} f(u_h)(v_h)_x dx + \hat{f}_{j+\frac{1}{2}} v_h(x^-_{j+\frac{1}{2}}) - \hat{f}_{j-\frac{1}{2}} v_h(x^+_{j-\frac{1}{2}}) = 0. \tag{9}$$

Here, $\hat{f}_{j+\frac{1}{2}} = \hat{f}(u_h(x^-_{j+\frac{1}{2}}, t), u_h(x^+_{j+\frac{1}{2}}, t))$ is the standard numerical flux, which is a single valued function defined at the cell interfaces and depends on the values of u_h from both sides of the interface. It can be chosen as a monotone flux in the scalar case and an exact or approximate Riemann solver for the system case.

For the two-dimensional conservation law (2), Equation (9) becomes

$$\int_{V_j} (u_h)_t v_h dxdy - \int_{V_j} \mathbf{F}(u_h) \cdot \nabla v_h dxdy + \int_{\partial V_j} \widehat{F}^n v_h ds = 0, \tag{10}$$

where $\mathbf{F} = (f, g)$, $\mathbf{n}$ is the outward unit normal vector of the cell boundary ∂V_j. $\widehat{F}^n = \widehat{F}^n(u_h^-, u_h^+, \mathbf{n})$ is the numerical flux along the normal direction of the cell boundary, consistent with $\mathbf{F} \cdot \mathbf{n}$. Here u_h^- and u_h^+ are the values of u_h inside the cell V_j and outside the cell V_j.

For the time discretization, we use a class of high order nonlinearly stable Runge-Kutta methods [23, 24, 44, 47]. They are convex combinations of first order forward Euler steps. The most popular scheme in this class is the following third order Runge-Kutta method for solving

$$u_t = L(u, t)$$

where $L(u,t)$ is a spatial discretization operator:

$$\begin{aligned} u^{(1)} &= u^n + \Delta t L(u^n, t^n), \\ u^{(2)} &= \frac{3}{4}u^n + \frac{1}{4}u^{(1)} + \frac{1}{4}\Delta t L(u^{(1)}, t^n + \Delta t), \\ u^{n+1} &= \frac{1}{3}u^n + \frac{2}{3}u^{(2)} + \frac{2}{3}\Delta t L(u^{(2)}, t^n + \frac{1}{2}\Delta t). \end{aligned} \tag{11}$$

3.2 Properties of our numerical method on cells generated from point clouds

As mentioned in the introduction, we first need to admit that our method does have the issue of relatively high computational cost. Besides the cost of generating the mesh (which is a one-time start-up cost for time dependent simulations), the main cost will be the computation of residues, i.e., cell and edge integrals in the DG formulations (9) and (10). For linear problems, if we write the solution as a combination of basis functions in each cell V_j and denote the vector of coefficients as u_j, then the DG formulations can be written in the form of $\frac{du_j}{dt} = B_{j,j}u_j + \sum_{i\in S_j} B_{j,i}u_i$, where S_j is the set of indices of immediate neighboring cells of the cell V_j (sharing an edge with V_j), and $B_{j,i}$ are small $K \times K$ matrices depending only on the mesh and the basis chosen. In this case, we can pre-compute and store the matrices $B_{j,i}$ at the beginning, and use matrix vector multiplications to compute the residue, so the cost would be comparable to that of the DG method for triangular meshes. For nonlinear problems with quadratic nonlinearities, such as Burgers equations, incompressible Navier-Stokes equations etc., the DG formulations can be written in the form of $\frac{du_j^\ell}{dt} = (u_j)^T B_{j,j,\ell}u_j + \sum_{i\in S_j}(u_i)^T B_{j,i,\ell}u_i$, where u_j^ℓ is the ℓ-th component of the vector u_j, S_j is still defined as above, and $B_{j,i,\ell}$ are small $K\times K$ matrices depending only on the mesh and the basis chosen. In this case, we can again pre-compute and store the matrices $B_{j,i,\ell}$ at the beginning, and use matrix vector multiplications to compute the residue, so the cost would be again comparable to that of the DG method for triangular meshes. For some of the general nonlinearities, such as the compressible Euler equations, we can use similar ideas such as the quadrature-free DG methodology

[1] to obtain similar efficient implementation. However, for most general nonlinear cases, the computation of the residual will be more costly, as quadrature rules need be used to compute the integrals in polygons with many edges. If we decompose the polygon into the union of several triangles and then use triangular quadrature rules on each of them, the computational cost would be proportional to the number of sides of the polygon.

Despite the computational cost issue, due to the good properties of the DG method, our method shares these same good properties. By taking $v_h = 1$, we can prove that the adopted method is locally conservative, which is a very important property for solving conservation laws, especially for non-smooth solutions. Jiang and Shu [29] have proven a cell entropy inequality for the one and two dimensional semi-discrete DG method for the square entropy, which implies that the numerical solutions, if convergent, will converge to an entropy solution, at least for the convex case. The entropy inequality also holds for DG solutions to symmetric systems [27]. Also, we have the L^2 stability for periodic or compactly supported boundary conditions. There are also some analysis of the stability of fully discretized DG schemes [52]. Regarding the accuracy issue, there are many works devoted to error estimates [30, 41, 50, 51]. Generally speaking, the L^2 error estimate of at least $O(h^{k+\frac{1}{2}})$ order in space can be proved on arbitrary meshes, where h in our context is the size of the cells V_j, which is related to the maximum distance between two adjacent points in the point cloud. The optimal error estimate $O(h^{k+1})$ can be observed in most situations and can be proved in many cases. For more detailed introduction to the DG method, we refer to [9, 26, 46].

Note that although the DG method for two-dimensional conservation laws is usually used on quadrilateral or triangular meshes, proofs of the entropy inequality and the L^2 stability are independent of the shape of cells. Hence, these results are still valid for our polygonal cells. For the implementation of the DG method, usually L^2 projection of the initial condition is used. Since we only know point values of the initial solution in the point cloud, we use polynomial interpolation or fitting instead of the L^2 projection.

This difference does not affect the proof of the L^2 error estimate, as long as the initial order of accuracy $\|u(\cdot,0)-u_h(\cdot,0)\| \leqslant Ch^{k+1}$ still holds for the polynomial interpolation or fitting, which is valid for our algorithm of forming the cells, since we control the condition number of the matrix for interpolation or fitting. Also, the L^2 error estimate of the order $O(h^{k+\frac{1}{2}})$ does not depend on the shape of the cells for DG methods, as long as they are regular (namely the ratio of the radius of the circumscribed circle over that of the inscribed circle of any cell stays bounded during refinement), hence we also have the $O(h^{k+\frac{1}{2}})$ error estimate provided our mesh is regular, which is again a reasonable assumption for our algorithm of forming the cells. In our numerical simulation, we do observe optimal order of accuracy as will be shown in the next section.

4 Numerical examples

In this section, we provide numerical experiments to demonstrate the performance of our mesh generating algorithm and the numerical scheme used. We show some numerical examples with smooth initial conditions in Section 4.1 and test the order of accuracy. In Section 4.2, we show some examples with discontinuous initial conditions.

In most numerical experiments, we test both uniform point clouds and random point clouds. For the former one, we use N (or $N \times N$ for the two-dimensional case) uniform points such that the resulting Voronoi diagram is a uniform decomposition of the computational domain. For the latter one, we also use N (or $N \times N$ for the two-dimensional case) number of points, but each point (or inner point for the two-dimensional case) in the cloud is randomly generated that satisfies a uniform distribution in the computational domain. For the two-dimensional case, the outmost nodes are set to be the same as that in the uniform point cloud, in order to impose periodic boundary conditions. Note that our mesh refinement is unstructured, that is, the generations of random points are independent with the refinement of N.

Each point in the cloud has an index. For the one-dimensional case, we just rank the

points from left to right. For the two-dimensional case with uniform points, we use the trivial way of indexing as shown in Figure 3. For two-dimensional randomly distributed points, we will compare different indexing of the points in Example 4. It appears that the performance of our algorithm is not sensitive to the specific ranking of the points. Therefore, we just maintain the indexing when we generate the points and thus eliminate the cost of ranking points. Since each point is randomly generated, the indexing is totally arbitrary.

In the procedure of grouping Voronoi regions into cells, we need to bound the condition number of the matrix A by a threshold value δ. Note that a large value of δ may lead to poor numerical interpolations. However, if δ is chosen too small, then it becomes harder to establish a cell that satisfies the condition number limit and more points may be added to neighboring existing cells, especially for highly inhomogeneous points. In general, the condition number for the matrix with higher order is larger. It is not easy to find a proper δ for all different orders of schemes (we will give some results to explain it in Example 1). In all numerical examples, we take δ as 100, 200 and 1000 for the one-dimensional second, third and fourth order schemes, respectively, and set it to be 100, 1000 and 3000 for two-dimensional second, third and fourth order schemes, respectively.

We use the third order TVD Runge-Kutta method for the time discretization. Second, third and fourth order DG schemes are tested in all examples. Since the time discretization is only third order accurate, we take $\Delta t \sim \Delta x^{4/3}$ to obtain fourth order accurate results for accuracy test examples.

We use the upwind numerical flux for linear numerical examples. In the one-dimensional case with $f(u) = au$, where a is a constant, we take

$$\hat{f}_{j+\frac{1}{2}}(u^-, u^+) = \begin{cases} au^-, & if\ a \geqslant 0, \\ au^+, & if\ a < 0. \end{cases} \tag{12}$$

In the two-dimensional linear case with $\mathbf{F}(u) = (au, bu)$, where a and b are constants,

we use

$$\widehat{F}^n(u^-, u^+, \mathbf{n}) = \begin{cases} \mathbf{F}(u^-) \cdot \mathbf{n}, & if\ (a,b) \cdot \mathbf{n} \geqslant 0, \\ \mathbf{F}(u^+) \cdot \mathbf{n}, & if\ (a,b) \cdot \mathbf{n} < 0. \end{cases} \tag{13}$$

For nonlinear numerical examples, we chose the Lax-Friedrichs flux. In the one-dimensional case, the flux is taken as

$$\hat{f}_{j+\frac{1}{2}}(u^-, u^+) = \frac{1}{2}[f(u^-) + f(u^+) - \alpha(u^+ - u^-)], \tag{14}$$

where $\alpha = \max_u |f'(u)|$. For the two-dimensional case, it becomes

$$\widehat{F}^n(u^-, u^+, \mathbf{n}) = \frac{1}{2}[\mathbf{F}(u^-) \cdot \mathbf{n} + \mathbf{F}(u^+) \cdot \mathbf{n} - \alpha(u^+ - u^-)], \tag{15}$$

where $\alpha = \max_{u,\mathbf{n}} |\mathbf{F}'(u) \cdot \mathbf{n}|$.

In each accuracy test example, we show two types of error tables. In the first type, we measure the standard numerical error of the piecewise polynomials on the entire computational region as in the traditional DG method. Since the initial data are only given on the point cloud and we care about the values on these points for the later time, we also show another error table measuring the numerical error on the points from the point cloud. For the L^∞ norm, we compute the maximum absolute value of the error on these points. For the L^1 norm, we multiply the absolute value of the error on each point with the area of the corresponding Voronoi region, add them together and divide the result by the area of the entire domain. For the one dimensional figures, we plot the numerical solutions on the given points in the point cloud. For the two dimensional cases, we divide the computational domain by a triangulation with the given points as the vertexes and thus plot the values on these points.

4.1 Smooth initial conditions

Example 1. We first test the performance of our method on the one-dimensional linear scalar problem

$$u_t + u_x = 0, \qquad 0 \leqslant x \leqslant 2\pi, \tag{16}$$

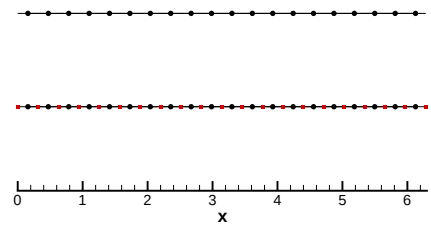
with the initial condition $u(x,0) = \sin(x)$ and a 2π-periodic boundary condition. The exact solution is $u(x,t) = \sin(x-t)$. Figure 4(a) and Figure 4(b) show the Voronoi diagrams generated by the uniform point cloud and the random point cloud with $N = 20$, respectively. We use black circles to denote the given points and use red squares to denote the boundary points of the Voronoi lines. Figures 4(c) to 4(h) show the resulting meshes by using the designed grouping algorithm with respect to different orders of the solution space. Numerical errors at $t = 2\pi$ are listed in Table 1 and Table 2. We show the results with uniform nodes in the left column and the results with random nodes in the right column. We can see that the order of accuracy for the random point cloud fluctuates, especially when $k = 2$, due to the highly inhomogeneity of the points. However, the average order of accuracy in the L^1 norm (measured by least square) is close to the optimal $(k+1)$-th order.

As we have already mentioned, we take δ as 100, 200 and 1000 for the one-dimensional second, third and fourth order schemes, respectively. In this numerical example, we show that δ can not be too large nor too small. If we take $\delta = 100$ for P^3, then we are unable to find any cell that satisfies the condition number limit for the random points. In Figure 5, we show the mesh decompositions by taking $\delta = 1000$ for P^2 and $\delta = 200$ for P^3, respectively. The resulting numerical errors are shown in Tables 3 and 4. We can see that if we take $\delta = 1000$ for P^2, then the mesh contains cells with poor interpolation property. If we take $\delta = 100$ for P^3, the first cell contains a lot of points and we need to approximate the initial data by using the least squares method. For both cases, the order of accuracy fluctuates a lot.

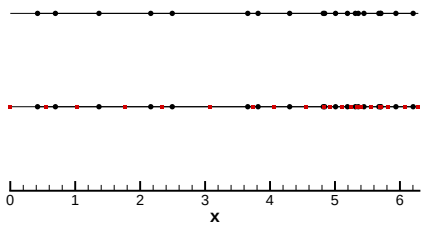
Example 2. We consider the one-dimensional Burgers equation

$$u_t + \left(\frac{u^2}{2}\right)_x = 0, \qquad 0 \leqslant x \leqslant 2\pi, \tag{17}$$

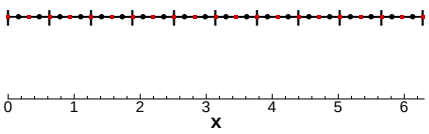
with the initial condition $u(x,0) = 0.5 + \sin(x)$ and periodic boundary conditions. We use the same point clouds as in the last example. Numerical errors at $t = 0.25$ when



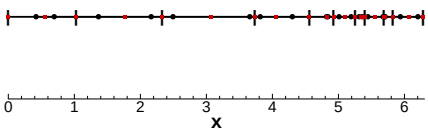
(a) uniform, Voronoi diagram



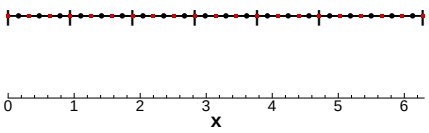
(b) random, Voronoi diagram



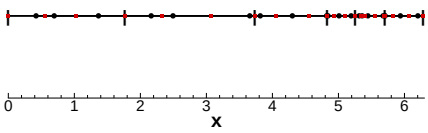
(c) uniform, P^1



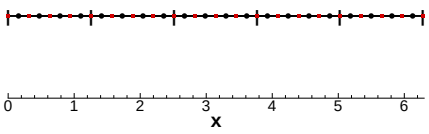
(d) random, P^1



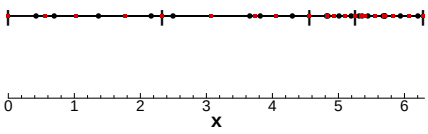
(e) uniform, P^2



(f) random, P^2



(g) uniform, P^3



(h) random, P^3

Figure 4: **Mesh decompositions of the one-dimensional domain** $[0, 2\pi]$**,** $N = 20$

Table 1: 1D linear equation with $u(x,0) = \sin(x)$ at $t = 2\pi$. Error on the whole domain.

N	uniform points				random points			
	L^1 norm	order	L^∞ norm	order	L^1 norm	order	L^∞ norm	order
	$k = 1$							
20	1.58E-02	–	3.43E-02	–	6.67E-02	–	1.46E-01	–
40	3.48E-03	2.18	1.15E-02	1.57	2.03E-02	1.72	8.86E-02	0.72
80	8.18E-04	2.09	3.27E-03	1.82	3.03E-03	2.75	1.64E-02	2.43
160	1.97E-04	2.05	8.63E-04	1.92	7.45E-04	2.02	8.07E-03	1.02
320	4.83E-05	2.03	2.21E-04	1.96	1.53E-04	2.29	1.97E-03	2.04
average	–	2.09	–	1.83	–	2.23	–	1.59
	$k = 2$							
20	5.40E-03	–	3.13E-02	–	1.52E-02	–	8.31E-02	–
40	3.45E-04	3.97	2.57E-03	3.61	1.24E-03	3.62	6.14E-03	3.76
80	4.79E-05	2.85	6.45E-04	1.99	1.79E-04	2.79	3.23E-03	0.93
160	4.71E-06	3.35	4.09E-05	3.98	1.65E-05	3.43	1.01E-04	5.00
320	6.08E-07	2.95	1.01E-05	2.02	2.84E-06	2.54	5.17E-05	0.97
average	–	3.24	–	2.92	–	3.10	–	2.72
	$k = 3$							
20	6.34E-04	–	1.91E-03	–	4.48E-03	–	1.76E-02	–
40	3.89E-05	4.03	1.18E-04	4.02	2.47E-04	4.18	1.48E-03	3.57
80	2.43E-06	4.00	7.45E-06	3.99	3.36E-05	2.88	1.85E-04	3.00
160	1.51E-07	4.01	4.71E-07	3.98	1.34E-06	4.65	8.65E-06	4.42
320	9.45E-09	4.00	2.95E-08	4.00	8.76E-08	3.93	1.27E-06	2.77
average	–	4.01	–	3.99	–	3.88	–	3.49

the solution is smooth are listed in Tables 5 and 6. Again, we can see that although the order of accuracy for the random point clouds fluctuates, the average order of accuracy (measured by least square), at least in the L^1 norm, is close to the optimal $(k+1)$-th

(a) random, P^2 with $\delta = 1000$ (b) random, P^3 with $\delta = 200$

Figure 5: **Mesh decompositions with inappropriate choices of δ**

Table 2: 1D linear equation with $u(x,0) = \sin(x)$ at $t = 2\pi$. Error on the given points.

N	uniform points				random points			
	L^1 norm	order	L^∞ norm	order	L^1 norm	order	L^∞ norm	order
	$k = 1$							
20	1.28E-02	–	3.44E-02	–	6.87E-02	–	1.31E-01	–
40	2.57E-03	2.32	6.72E-03	2.36	1.91E-02	1.85	6.23E-02	1.08
80	6.52E-04	1.98	1.36E-03	2.30	2.70E-03	2.83	1.10E-02	2.51
160	1.63E-04	2.00	2.99E-04	2.19	7.03E-04	1.94	4.34E-03	1.34
320	4.09E-05	2.00	6.95E-05	2.10	1.48E-04	2.25	1.77E-03	1.29
average	–	2.06	–	2.24	–	2.25	–	1.63
	$k = 2$							
20	4.85E-03	–	1.60E-02	–	1.23E-02	–	3.85E-02	–
40	2.42E-04	4.33	1.37E-03	3.55	9.55E-04	3.69	3.84E-03	3.33
80	3.64E-05	2.73	3.09E-04	2.15	1.50E-04	2.67	9.44E-04	2.02
160	3.15E-06	3.53	2.29E-05	3.76	1.51E-05	3.31	7.13E-05	3.73
320	4.20E-07	2.91	5.14E-06	2.15	2.46E-06	2.62	4.14E-05	0.78
average	–	3.33	–	2.91	–	3.06	–	2.55
	$k = 3$							
20	4.85E-04	–	1.40E-03	–	3.76E-03	–	8.02E-03	–
40	2.79E-05	4.12	9.65E-05	3.86	1.99E-04	4.24	1.18E-03	2.77
80	1.71E-06	4.03	6.04E-06	4.00	2.84E-05	2.80	1.57E-04	2.91
160	1.06E-07	4.01	3.80E-07	3.99	1.29E-06	4.46	1.20E-05	3.72
320	6.65E-09	4.00	2.38E-08	4.00	6.79E-08	4.25	8.68E-07	3.79
average	–	4.03	–	3.97	–	3.88	–	3.30

order. In Figure 6, we show the numerical results using 80 cells at $t = 1.5$, when a shock has already appeared in the solution. We can see that the results are quite good but there are some oscillations near the discontinuity. This is expected as we have not used any limiters to control these oscillations. The design and application of suitable limiters for our method will be studied in our future work, as discussed in the concluding remarks section.

Table 3: 1D linear equation with inappropriate choices of δ. Error on the whole domain.

N	P^2 with $\delta = 1000$				P^3 with $\delta = 200$			
	L^1 norm	order	L^∞ norm	order	L^1 norm	order	L^∞ norm	order
20	1.52E-02	–	8.31E-02	–	6.38E-02	–	9.81E-02	–
40	9.53E-04	4.00	8.53E-03	3.28	4.29E-04	7.22	2.74E-03	5.16
80	5.81E-04	0.71	4.73E-03	0.85	3.82E-05	3.49	1.87E-04	3.87
160	1.58E-05	5.20	1.02E-04	5.53	2.13E-05	0.85	2.76E-04	-0.56
320	4.27E-06	1.89	8.01E-05	0.35	1.56E-07	7.09	1.78E-06	7.28
average	–	2.95	–	2.64	–	4.16	–	3.48

Table 4: 1D linear equation with inappropriate choices of δ. Error on the given points.

N	P^2 with $\delta = 1000$				P^3 with $\delta = 200$			
	L^1 norm	order	L^∞ norm	order	L^1 norm	order	L^∞ norm	order
20	1.21E-02	–	3.82E-02	–	6.18E-02	–	1.25E-01	–
40	1.14E-03	3.40	4.58E-03	3.06	2.81E-04	7.78	9.28E-04	7.07
80	5.75E-04	0.99	5.63E-03	-0.30	3.09E-05	3.19	1.59E-04	2.55
160	1.41E-05	5.35	7.31E-05	6.27	1.98E-05	0.64	3.13E-04	-0.98
320	3.91E-06	1.84	7.43E-05	-0.02	1.34E-07	7.20	2.26E-06	7.11
average	–	2.95	–	2.40	–	4.15	–	3.31

Example 3. Consider the one-dimensional Euler system of compressible gas dynamics

$$\mathbf{u}_t + \mathbf{f}(\mathbf{u})_x = 0,$$

$$\mathbf{u} = \begin{pmatrix} \rho \\ m \\ E \end{pmatrix}, \qquad \mathbf{f}(\mathbf{u}) = \begin{pmatrix} m \\ \rho v^2 + p \\ v(E+p) \end{pmatrix}. \tag{18}$$

Here ρ is the density, v is the velocity, $m = \rho v$ is the momentum, E is the total energy, and p is the pressure, with the equation of state

$$p(\mathbf{u}) = (\gamma - 1)(E - \frac{1}{2}\rho v^2); \tag{19}$$

$\gamma = 1.4$ for the air. The initial condition is set to be $\rho(x, 0) = 1 + 0.2\sin(x)$, $v(x, 0) = 1$, $p(x, 0) = 1$, with a 2π-periodic boundary condition. The exact solution is $\rho(x, t) = 1 + 0.2\sin(x - t)$, $v(x, t) = 1$ and $p(x, t) = 1$. We show the numerical errors for the density at $t = 2\pi$ in Tables 7 and 8. We can see that the error in the L^1 norm (in average) can reach the optimal $(k + 1)$-th order of accuracy.

Table 5: 1D Burgers equation with $u(x,0) = 0.5 + \sin(x)$ at $t = 0.25$. Error on the whole domain.

N	uniform points				random points			
	L^1 norm	order	L^∞ norm	order	L^1 norm	order	L^∞ norm	order
	$k = 1$							
20	1.07E-02	–	5.65E-02	–	3.07E-02	–	1.76E-01	–
40	2.58E-03	2.05	1.59E-02	1.83	1.56E-02	0.97	1.03E-01	0.77
80	6.32E-04	2.03	4.19E-03	1.93	2.57E-03	2.60	1.74E-02	2.57
160	1.57E-04	2.01	1.07E-03	1.96	7.88E-04	1.70	9.24E-03	0.91
320	3.93E-05	2.00	2.72E-04	1.98	1.74E-04	2.18	1.57E-03	2.56
average	–	2.02	–	1.93	–	1.92	–	1.71
	$k = 2$							
20	4.67E-03	–	1.30E-02	–	1.44E-02	–	7.93E-02	–
40	4.69E-04	3.31	2.29E-03	2.51	1.38E-03	3.38	8.42E-03	3.23
80	7.66E-05	2.62	3.83E-04	2.58	2.49E-04	2.47	1.74E-03	2.28
160	9.99E-06	2.94	6.84E-05	2.49	3.44E-05	2.86	2.53E-04	2.78
320	1.48E-06	2.76	1.15E-05	2.58	4.85E-06	2.82	3.89E-05	2.70
average	–	2.88	–	2.54	–	2.84	–	2.70
	$k = 3$							
20	4.96E-04	–	1.59E-03	–	5.32E-03	–	1.20E-02	–
40	5.64E-05	3.13	2.81E-04	2.50	3.12E-04	4.09	1.83E-03	2.72
80	3.00E-06	4.23	1.76E-05	4.00	3.08E-05	3.34	1.96E-04	3.23
160	1.94E-07	3.95	1.23E-06	3.83	1.73E-06	4.15	1.40E-05	3.81
320	1.18E-08	4.03	7.89E-08	3.96	1.23E-07	3.82	1.33E-06	3.39
average	–	3.89	–	3.64	–	3.83	–	3.33

Example 4. From now on, we consider two-dimensional cases. Let us first consider the two-dimensional linear equation

$$u_t + u_x - 2u_y = 0, \qquad 0 \le x, y \le 2\pi, \tag{20}$$

with the initial condition $u(x, y, 0) = \sin(x + y)$ and a 2π-periodic boundary condition. Figure 7 shows 20×20 uniform and random point clouds and their corresponding Voronoi diagrams. Figure 8 shows mesh subdivisions of these points for different orders of schemes. Here we use red lines to denote Voronoi edges and use black lines to denote cell boundaries of the mesh. Numerical errors at $t = 2\pi$ are listed in Tables 9 and 10. Again, we can see that the L^1 norm of the error (in average) can reach the optimal

Table 6: 1D Burgers equation with $u(x,0) = 0.5 + \sin(x)$ at $t = 0.25$. Error on the given points.

N	uniform points				random points			
	L^1 norm	order	L^∞ norm	order	L^1 norm	order	L^∞ norm	order
	$k = 1$							
20	6.48E-03	–	1.65E-02	–	2.09E-02	–	5.89E-02	–
40	1.54E-03	2.08	4.72E-03	1.80	1.69E-02	0.31	1.00E-01	-0.76
80	3.73E-04	2.04	1.25E-03	1.92	2.30E-03	2.88	1.18E-02	3.08
160	9.29E-05	2.00	3.18E-04	1.97	6.81E-04	1.75	3.25E-03	1.86
320	2.32E-05	2.00	8.01E-05	1.99	1.77E-04	1.95	1.47E-03	1.14
average	–	2.03	–	1.93	–	1.84	–	1.56
	$k = 2$							
20	4.29E-03	–	1.08E-02	–	7.43E-03	–	2.24E-02	–
40	4.25E-04	3.33	1.58E-03	2.77	1.58E-03	2.24	6.75E-03	1.73
80	6.18E-05	2.78	2.39E-04	2.73	2.25E-04	2.81	1.03E-03	2.71
160	7.62E-06	3.02	3.76E-05	2.67	2.93E-05	2.94	1.64E-04	2.66
320	1.09E-06	2.81	6.61E-06	2.51	5.03E-06	2.54	3.84E-05	2.09
average	–	2.97	–	2.67	–	2.68	–	2.37
	$k = 3$							
20	4.46E-04	–	1.79E-03	–	5.25E-03	–	1.21E-02	–
40	4.87E-05	3.19	2.72E-04	2.72	3.64E-04	3.85	2.26E-03	2.42
80	2.46E-06	4.31	1.34E-05	4.34	3.24E-05	3.49	1.77E-04	3.68
160	1.60E-07	3.95	8.86E-07	3.92	1.76E-06	4.20	1.34E-05	3.72
320	9.62E-09	4.05	5.41E-08	4.03	1.21E-07	3.86	1.33E-06	3.34
average	–	3.93	–	3.83	–	3.85	–	3.37

$(k+1)$-th order of accuracy.

Note that the results on random points in Tables 9 and 10 are obtained by just using the index of each point when it is generated. We now test different rules of sort these random points. For Rule 1, we rank the random points by the value of $x + y$ in increasing order. If two points have the same value of $x + y$, the point with smaller x has the priority. For Rule 2, we first rank the points by y and then use the value of x to rank the points with the same value of y. The mesh decompositions by using different rules are shown in Figure 9. Numerical errors at $t = 2\pi$ are listed in Tables 11 and 12. It appears that the results are not sensitive to the specific ranking of the random points.

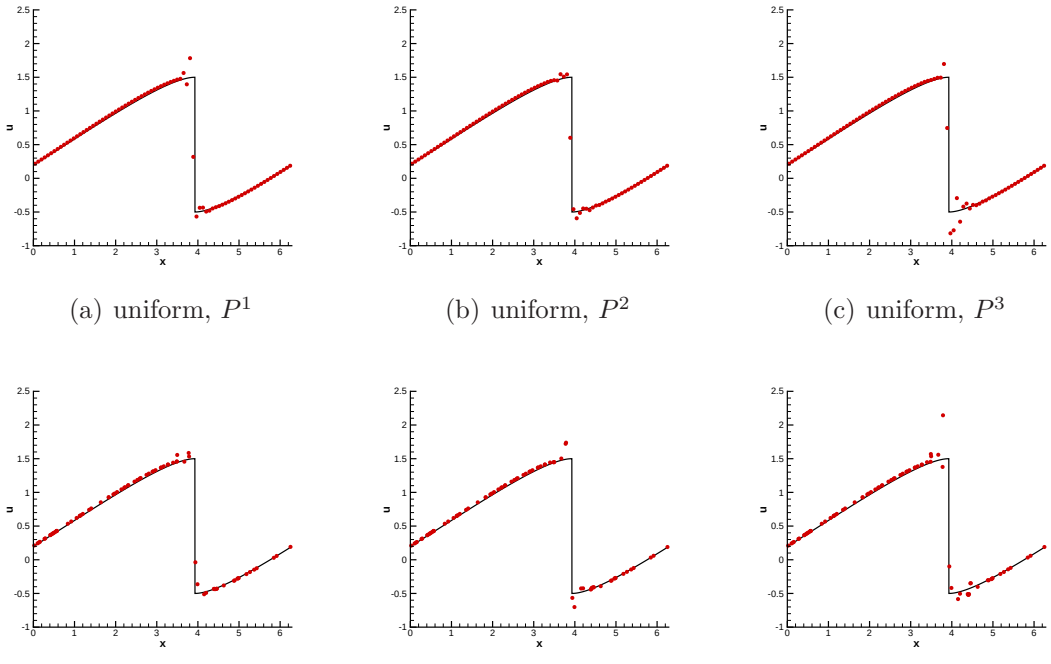
Therefore, we will simply take the ranking by using the index of each point when it is generated in the remaining examples to save computational cost.

Example 5. We consider the two-dimensional Burgers equation

$$u_t + \left(\frac{u^2}{2}\right)_x + \left(\frac{u^2}{2}\right)_y = 0, \qquad 0 \leqslant x, y \leqslant 2\pi, \tag{21}$$

with the initial condition $u(x, y, 0) = 0.5 + \sin(x + y)$ and periodic boundary conditions. We use the same mesh as in Example 4. Numerical errors at $t = 0.25$ when the solution is smooth are listed in Tables 13 and 14. We can see that the L^1 norm of the error (in average) is close to the optimal order of accuracy.

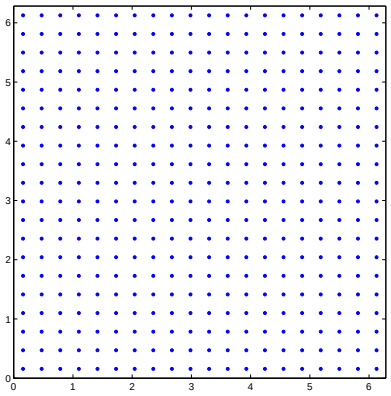
In this example, we investigate the computational times (in seconds) for different parts of our algorithm with $N = 160$. In Table 15, "Voronoi" represents the cost of Voronoi decomposition and collecting relevant decomposition information. "Grouping" represents the cost of grouping Voronoi regions into cells. "Initialization" represents the cost of computing relevant matrices needed in the DG method and approximating the



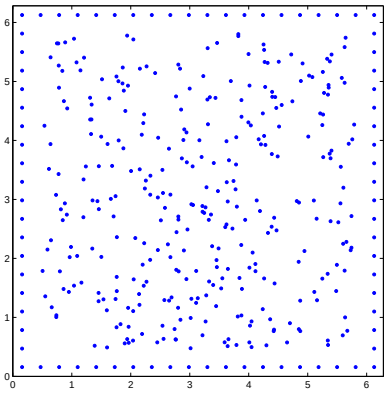
(a) uniform, P^1 (b) uniform, P^2 (c) uniform, P^3

(d) random, P^1 (e) random, P^2 (f) random, P^3

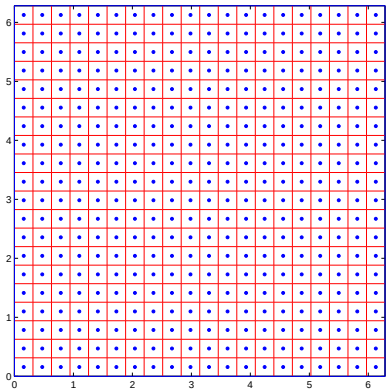
Figure 6: **1D Burgers problem at** $T = 1.5$**,** $N = 80$



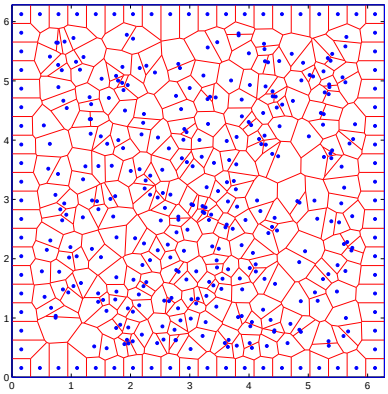
(a) uniform, point cloud



(b) random, point cloud

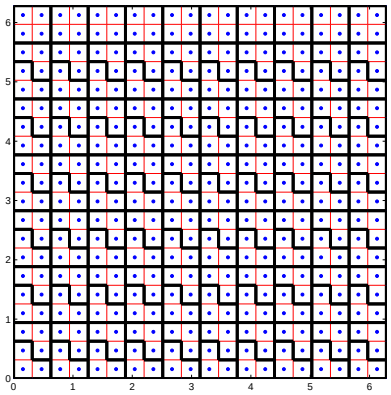


(c) uniform, Voronoi diagram

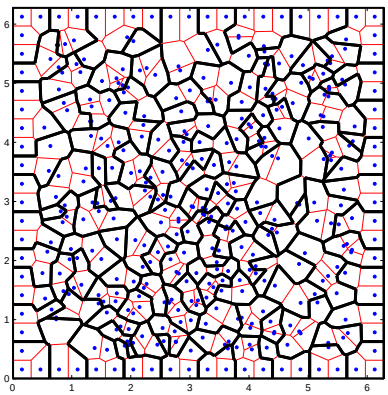


(d) random, Voronoi diagram

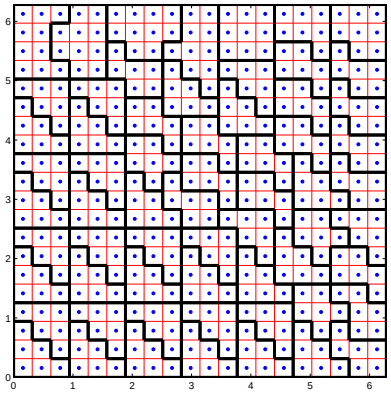
Figure 7: **Point clouds and Voronoi diagrams in the two-dimensional domain** $[0, 2\pi] \times [0, 2\pi]$**,** $N_x = N_y = 20$



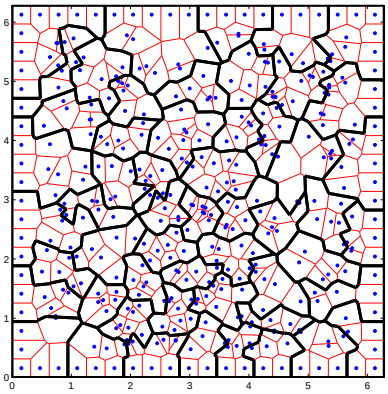
(a) uniform, P^1



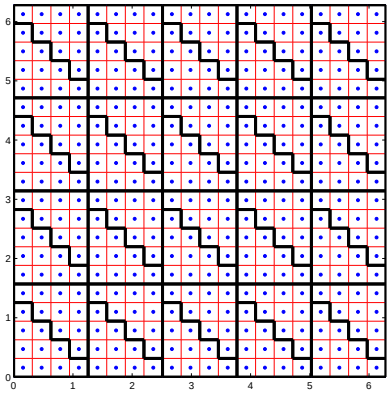
(b) random, P^1



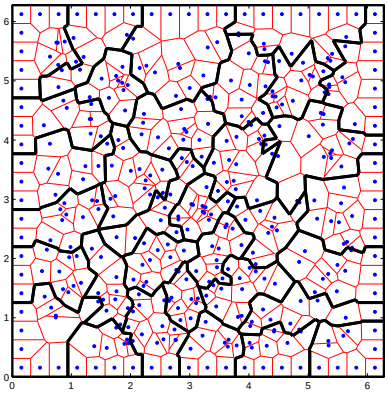
(c) uniform, P^2



(d) random, P^2

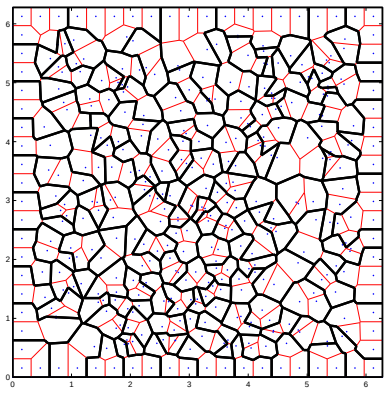


(e) uniform, P^3

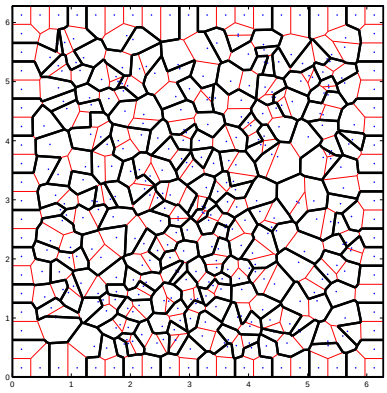


(f) random, P^3

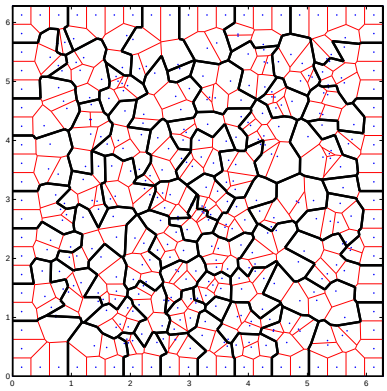
Figure 8: **Mesh decompositions of the two-dimensional domain** $[0, 2\pi] \times [0, 2\pi]$**,** $N_x = N_y = 20$



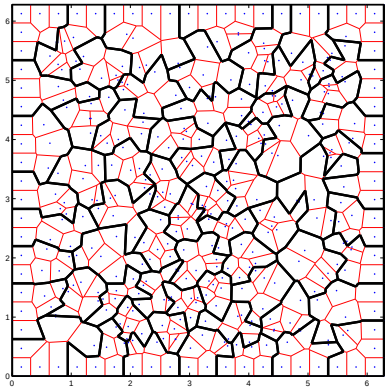
(a) Rule 1, P^1



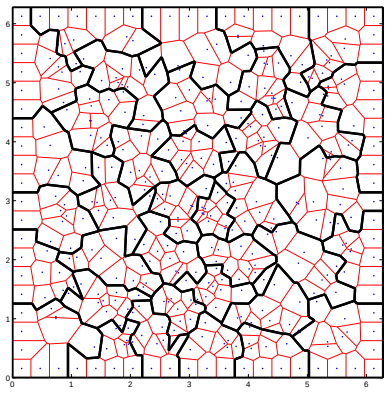
(b) Rule 2, P^1



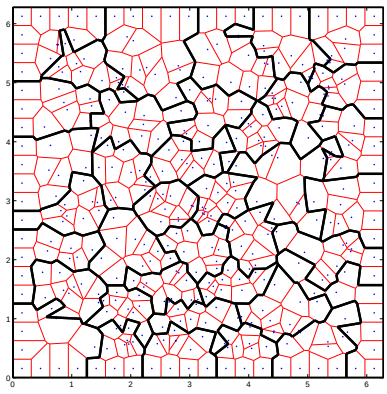
(c) Rule 1, P^2



(d) Rule 2, P^2



(e) Rule 1, P^3



(f) Rule 2, P^3

Figure 9: **Mesh decompositions of the two-dimensional domain** $[0, 2\pi] \times [0, 2\pi]$ **with different indexing rules,** $N_x = N_y = 20$

Table 7: 1D Euler equation with $\rho(x,0) = 1 + 0.2\sin(x)$, $v(x,0) = 1$ and $p(x,0) = 1$ at $t = 2\pi$. Error on the whole domain.

N	uniform points				random points			
	L^1 norm	order	L^∞ norm	order	L^1 norm	order	L^∞ norm	order
	$k = 1$							
20	2.00E-03	–	6.37E-03	–	1.26E-02	–	3.09E-02	–
40	4.54E-04	2.14	1.82E-03	1.81	3.81E-03	1.73	1.76E-02	0.81
80	1.11E-04	2.03	4.84E-04	1.91	5.42E-04	2.82	3.18E-03	2.47
160	2.77E-05	2.01	1.25E-04	1.96	1.46E-04	1.90	1.59E-03	1.01
320	6.90E-06	2.00	3.16E-05	1.98	3.05E-05	2.26	3.85E-04	2.04
average	–	2.04	–	1.92	–	2.21	–	1.61
	$k = 2$							
20	1.29E-03	–	6.75E-03	–	3.45E-03	–	1.73E-02	–
40	1.11E-04	3.54	6.38E-04	3.40	3.08E-04	3.49	1.52E-03	3.52
80	1.62E-05	2.78	1.49E-04	2.10	5.00E-05	2.62	6.60E-04	1.20
160	1.68E-06	3.27	1.08E-05	3.79	4.74E-06	3.40	2.67E-05	4.63
320	2.17E-07	2.96	2.33E-06	2.21	7.38E-07	2.68	1.07E-05	1.32
average	–	3.11	–	2.89	–	3.04	–	2.71
	$k = 3$							
20	7.79E-05	–	2.79E-04	–	1.16E-03	–	4.56E-03	–
40	4.35E-06	4.16	1.23E-05	4.51	5.58E-05	4.38	2.96E-04	3.94
80	2.57E-07	4.08	7.03E-07	4.13	7.65E-06	2.87	3.64E-05	3.03
160	1.56E-08	4.04	4.46E-08	3.98	2.73E-07	4.81	1.80E-06	4.34
320	9.72E-10	4.01	2.78E-09	4.00	1.80E-08	3.92	2.50E-07	2.85
average	–	4.07	–	4.13	–	3.96	–	3.57

discrete initial data with element-wise polynomials. "DG" represents the cost of marching in time using DG method. We can see that the procedure of Voronoi decomposition and collecting decomposition information costs some time. But it dose not depend on the order of the scheme and hence can be done just once for each point cloud. The costs for grouping and initialization are very small. Despite the Voronoi start-up cost, the majority cost is using DG to march in time, as mentioned in Section 3.2. Note that the DG cost becomes smaller as k increases because there are fewer cells. In general, the costs for random points are larger than the costs for uniform points.

Table 8: 1D Euler equation with $\rho(x,0) = 1 + 0.2\sin(x)$, $v(x,0) = 1$ and $p(x,0) = 1$ at $t = 2\pi$. Error on the given points.

N	uniform points				random points			
	L^1 norm	order	L^∞ norm	order	L^1 norm	order	L^∞ norm	order
	$k = 1$							
20	1.38E-03	–	3.71E-03	–	1.15E-02	–	2.48E-02	–
40	2.33E-04	2.57	6.07E-04	2.61	3.52E-03	1.71	1.19E-02	1.06
80	5.69E-05	2.04	1.19E-04	2.36	4.74E-04	2.89	1.84E-03	2.69
160	1.41E-05	2.01	2.58E-05	2.20	1.33E-04	1.84	8.29E-04	1.15
320	3.52E-06	2.00	5.99E-06	2.11	2.80E-05	2.24	3.45E-04	1.26
average	–	2.13	–	2.31	–	2.21	–	1.62
	$k = 2$							
20	1.09E-03	–	3.61E-03	–	2.73E-03	–	8.24E-03	–
40	8.40E-05	3.70	3.24E-04	3.48	2.68E-04	3.35	1.21E-03	2.76
80	1.24E-05	2.76	7.45E-05	2.12	4.76E-05	2.49	2.22E-04	2.45
160	1.19E-06	3.38	5.86E-06	3.67	4.49E-06	3.41	2.17E-05	3.35
320	1.54E-07	2.95	1.13E-06	2.37	6.77E-07	2.73	8.75E-06	1.31
average	–	3.17	–	2.91	–	2.99	–	2.56
	$k = 3$							
20	4.12E-05	–	1.17E-04	–	1.05E-03	–	1.92E-03	–
40	3.19E-06	3.69	8.04E-05	3.86	4.48E-05	4.55	2.23E-04	3.11
80	1.92E-07	4.06	5.42E-06	3.89	6.35E-06	2.82	3.17E-05	2.81
160	1.17E-08	4.03	3.30E-07	4.04	2.53E-07	4.65	1.96E-06	4.02
320	7.32E-10	4.00	2.06E-09	4.00	1.38E-08	4.19	1.74E-07	3.49
average	–	3.97	–	3.95	–	3.99	–	3.37

Example 6. Let us consider the two-dimensional Euler system which is given by

$$\mathbf{u}_t + \mathbf{f}(\mathbf{u})_x + \mathbf{g}(\mathbf{u})_y = 0,$$

$$\mathbf{u} = \begin{pmatrix} \rho \\ m \\ n \\ E \end{pmatrix}, \mathbf{f}(\mathbf{u}) = \begin{pmatrix} m \\ \rho u^2 + p \\ \rho uv \\ u(E+p) \end{pmatrix}, \mathbf{g}(\mathbf{u}) = \begin{pmatrix} n \\ \rho uv \\ \rho v^2 + p \\ v(E+p) \end{pmatrix}. \tag{22}$$

Here, ρ is the density, $(u,v)^T$ is the velocity vector, $m = \rho u$ and $n = \rho v$ are the momenta, E is the total energy, and p is the pressure, with the equation of state

$$p(\mathbf{u}) = (\gamma - 1)\big(E - \frac{1}{2}\rho(u^2 + v^2)\big). \tag{23}$$

The initial condition is set to be $\rho(x,y,0) = 1+0.2\sin(x+y)$, $u(x,y,0) = 0.7$, $v(x,y,0) =$

Table 9: 2D linear equation with $u(x, y, 0) = \sin(x + y)$ at $t = 2\pi$. Error on the whole domain.

$N_x \times N_y$	uniform points				random points			
	L^1 norm	order	L^∞ norm	order	L^1 norm	order	L^∞ norm	order
	$k = 1$							
20×20	4.38E-02	–	0.13	–	0.13	–	0.38	–
40×40	6.13E-03	2.84	4.02E-02	1.71	1.76E-02	2.89	0.12	1.68
80×80	1.17E-03	2.39	1.09E-02	1.89	3.60E-03	2.29	4.44E-02	1.41
160×160	2.72E-04	2.11	2.96E-03	1.88	8.00E-04	2.17	1.44E-02	1.62
320×320	6.74E-05	2.01	7.34E-04	2.01	2.07E-04	1.95	3.39E-03	2.09
average	–	2.32	–	1.87	–	2.30	–	1.66
	$k = 2$							
20×20	7.31E-03	–	0.11	–	2.34E-02	–	0.24	–
40×40	6.93E-04	3.40	1.84E-02	2.62	2.76E-03	3.08	3.46E-02	2.78
80×80	7.20E-05	3.27	2.04E-03	3.17	4.24E-04	2.70	8.08E-03	2.10
160×160	9.20E-06	2.97	2.58E-04	2.98	3.49E-05	3.60	1.05E-03	2.94
320×320	1.01E-06	3.19	3.42E-05	2.92	4.63E-06	2.91	1.02E-04	3.36
average	–	3.19	–	2.95	–	3.09	–	2.74
	$k = 3$							
20×20	1.09E-03	–	8.49E-03	–	9.33E-03	–	7.77E-02	–
40×40	6.72E-05	4.02	6.37E-04	3.74	4.48E-04	4.38	7.30E-03	3.41
80×80	4.06E-06	4.05	4.24E-05	3.91	3.16E-05	3.82	4.69E-04	3.96
160×160	2.50E-07	4.02	2.69E-06	3.98	1.71E-06	4.21	4.20E-05	3.48
320×320	1.56E-08	4.01	1.69E-07	3.99	1.08E-07	3.99	3.14E-06	3.74
average	–	4.03	–	3.91	–	4.08	–	3.66

0.3 and $p(x, y, 0) = 1$, $0 \le x, y \le 2\pi$. The boundary conditions are periodic. $\gamma = 1.4$ is used in the computation. The exact solution is $\rho(x, y, t) = 1 + 0.2\sin(x + y - t)$, $u(x, y, t) = 0.7$, $v(x, y, t) = 0.3$ and $p(x, y, t) = 1$. For this test case, we use the same mesh as in Example 4. Tables 16 and 17 show the L^1 and L^∞ errors and numerical orders of accuracy of the density at $t = 2\pi$. We can see that the L^1 norm of errors (in average) can reach the $(k + 1)$-th order of accuracy.

Example 7. Consider the two-dimensional vortex evolution problem, which is an idealized problem for the two-dimensional Euler equations [45]. The setup of this problem is: The mean flow is $\rho = 1$, $p = 1$ and $(u, v) = (1, 1)$ (diagonal flow). We add, to this

Table 10: 2D linear equation with $u(x, y, 0) = \sin(x + y)$ at $t = 2\pi$. Error on the given points.

$N_x \times N_y$	uniform points				random points			
	L^1 norm	order	L^∞ norm	order	L^1 norm	order	L^∞ norm	order
	$k = 1$							
20×20	4.92E-02	–	1.15E-01	–	1.32E-01	–	3.56E-01	–
40×40	6.04E-03	3.03	2.05E-02	2.49	1.74E-02	2.93	7.52E-02	2.24
80×80	7.45E-04	3.02	4.11E-03	2.32	3.26E-03	2.41	1.98E-02	1.92
160×160	1.24E-04	2.59	8.51E-04	2.27	6.93E-04	2.24	7.83E-03	1.34
320×320	2.71E-05	2.19	2.03E-04	2.06	1.78E-04	1.96	2.35E-03	1.74
average	–	2.73	–	2.29	–	2.37	–	1.78
	$k = 2$							
20×20	6.42E-03	–	3.09E-02	–	2.15E-02	–	8.57E-02	–
40×40	5.86E-04	3.45	5.33E-03	2.53	2.50E-03	3.10	1.75E-02	2.29
80×80	6.08E-05	3.27	5.65E-04	3.24	4.00E-04	2.65	3.66E-03	2.26
160×160	7.73E-06	2.97	8.92E-05	2.66	3.12E-05	3.68	3.17E-04	3.53
320×320	8.42E-07	3.20	9.15E-06	3.29	4.23E-06	2.88	4.49E-05	2.82
average	–	3.20	–	2.93	–	3.09	–	2.76
	$k = 3$							
20×20	9.76E-04	–	2.40E-03	–	8.78E-03	–	3.44E-02	–
40×40	5.85E-05	4.06	1.64E-04	3.87	4.25E-04	4.37	2.48E-03	3.80
80×80	3.50E-06	4.06	1.03E-05	4.00	2.97E-05	3.84	2.06E-04	3.59
160×160	2.15E-07	4.02	6.40E-07	4.00	1.62E-06	4.19	3.40E-05	2.60
320×320	1.34E-08	4.01	4.00E-08	4.00	1.02E-07	3.99	1.26E-06	4.76
average	–	4.04	–	3.97	–	4.08	–	3.57

mean flow, an isentropic vortex (perturbation in (u, v) and the temperature $T = \frac{p}{\rho}$, no perturbation in the entropy $S = \frac{p}{\rho^\gamma}$):

$$(\delta u, \delta v) = \frac{\epsilon}{2\pi} e^{0.5(1-t^2)}(-\bar{y}, \bar{x}), \qquad \delta T = -\frac{(\gamma - 1)\epsilon^2}{8\gamma\pi^2} e^{1-r^2}, \qquad \delta S = 0, \tag{24}$$

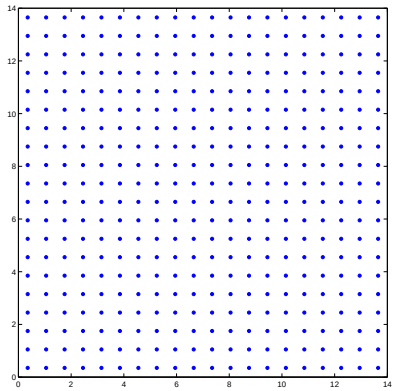
where $(\bar{x}, \bar{y}) = (x - 7, y - 7)$, $r^2 = \bar{x}^2 + \bar{y}^2$, and the vortex strength $\epsilon = 5$. The computational domain is taken as $[0, 14] \times [0, 14]$, extended periodically in both directions. It is clear that the exact solution of the Euler equation with the above initial and boundary conditions is just the passive convection of the vortex with the mean velocity. We show the point clouds and the corresponding Voronoi diagrams in Figure 10. Mesh decompositions are shown in Figure 11. Errors and orders of accuracy for the density at $t = 0.2$

Table 11: 2D linear equation with different indexing rules. Error on the whole domain.

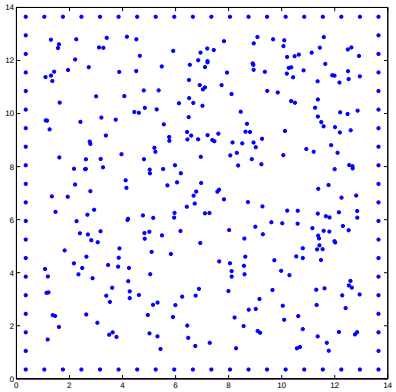
$N_x \times N_y$	Rule 1				Rule 2			
	L^1 norm	order	L^∞ norm	order	L^1 norm	order	L^∞ norm	order
	$k = 1$							
20×20	8.36E-02	–	0.31	–	0.11	–	0.31	–
40×40	1.16E-02	2.85	8.64E-02	1.84	1.48E-02	2.90	0.11	1.52
80×80	2.34E-03	2.31	2.77E-02	1.64	2.98E-03	2.32	2.99E-02	1.83
160×160	5.36E-04	2.12	6.88E-03	2.01	6.81E-04	2.13	8.81E-03	1.76
320×320	1.32E-04	2.03	2.70E-03	1.35	1.83E-04	1.89	2.90E-03	1.60
average	–	2.31	–	1.73	–	2.29	–	1.70
	$k = 2$							
20×20	1.94E-02	–	0.15	–	1.82E-02	–	0.12	–
40×40	1.52E-03	3.67	2.13E-02	2.82	2.42E-03	2.91	2.46E-02	2.23
80×80	2.02E-04	2.91	3.37E-03	2.66	2.25E-04	3.43	4.62E-03	2.41
160×160	2.51E-05	3.01	5.19E-04	2.70	2.91E-05	2.95	8.80E-04	2.39
320×320	3.30E-06	2.93	7.35E-05	2.82	3.29E-06	3.14	7.02E-05	3.65
average	–	3.10	–	2.74	–	3.12	–	2.62
	$k = 3$							
20×20	3.19E-03	–	4.74E-02	–	4.26E-03	–	5.37E-02	–
40×40	2.42E-04	3.72	4.17E-03	3.51	3.40E-04	3.65	4.24E-03	3.66
80×80	1.63E-05	3.89	3.00E-04	3.80	3.24E-05	3.39	4.25E-04	3.32
160×160	9.32E-07	4.13	2.03E-05	3.88	1.39E-06	4.54	3.37E-05	3.65
320×320	7.51E-08	3.63	2.47E-06	3.04	7.85E-08	4.15	2.65E-06	3.67
average	–	3.88	–	3.61	–	3.94	–	3.56

are shown in Tables 18 and 19. The L^1 norm of the error is close to the optimal order of accuracy.

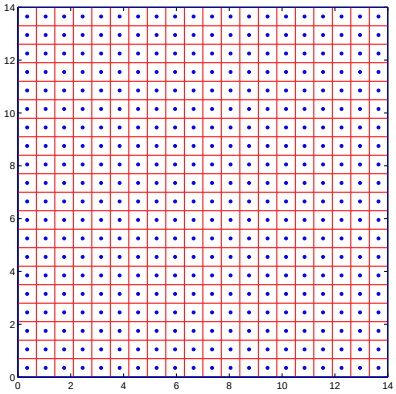
We can see, in the majority of numerical experiments, larger fluctuations of the order of convergence for the $k = 2$ case even for uniform points. This effect appears to be caused by the mesh decomposition. For $k = 2$, the mesh decomposition appears to be more irregular and there is a larger percentage of cells containing more than K points and hence we need to use the least squares method to fit the initial data on these cells, which may cause the fluctuation of the order of accuracy.



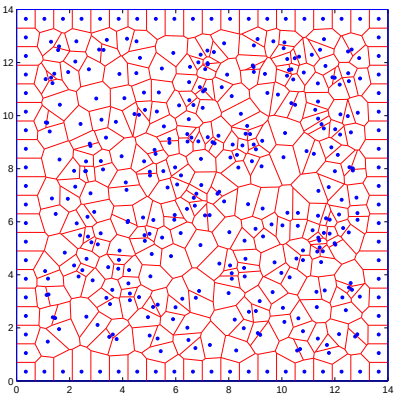
(a) uniform, point cloud



(b) random, point cloud

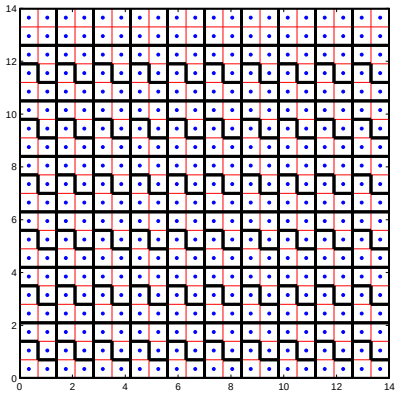


(c) uniform, Voronoi diagram

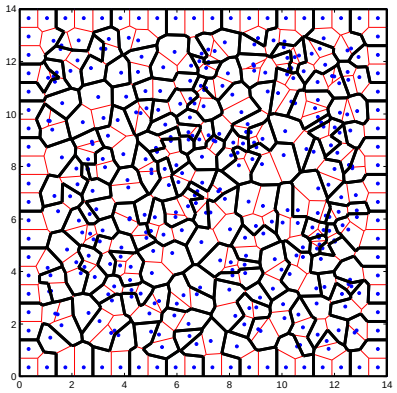


(d) random, Voronoi diagram

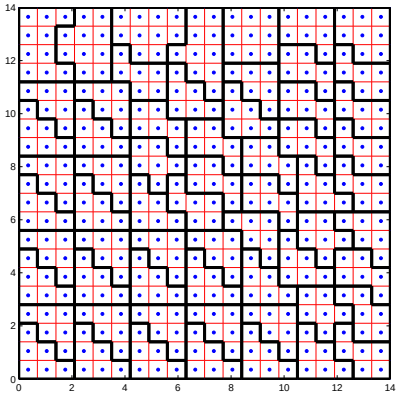
Figure 10: **Point clouds and Voronoi diagrams in the two-dimensional domain** $[0, 14] \times [0, 14]$**,** $N_x = N_y = 20$



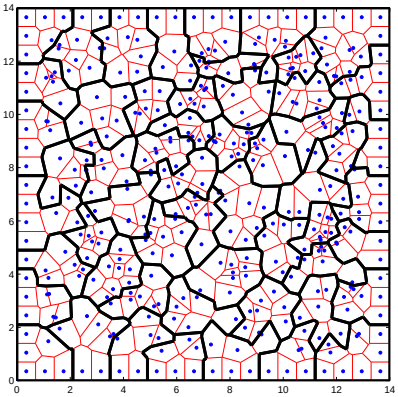
(a) uniform, P^1



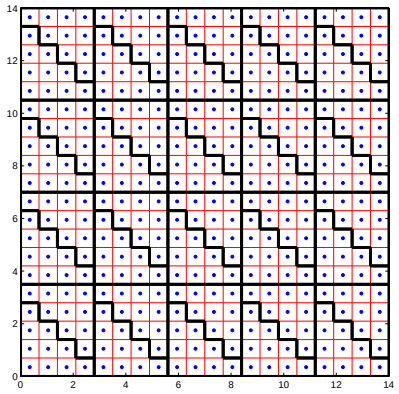
(b) random, P^1



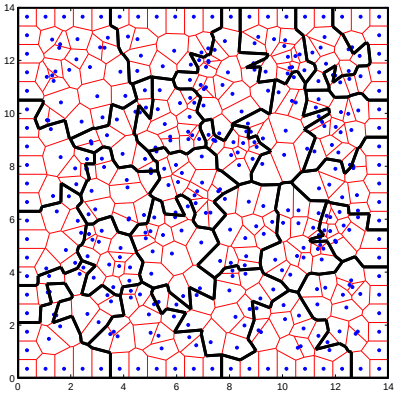
(c) uniform, P^2



(d) random, P^2



(e) uniform, P^3



(f) random, P^3

Figure 11: **Mesh decompositions of the two-dimensional domain** $[0, 14] \times [0, 14]$**,** $N_x = N_y = 20$

Table 12: 2D linear equation with different indexing rules. Error on the given points.

$N_x \times N_y$	Rule 1				Rule 2			
	L^1 norm	order	L^∞ norm	order	L^1 norm	order	L^∞ norm	order
	$k=1$							
20×20	8.85E-02	–	2.38E-01	–	1.15E-01	–	2.56E-01	–
40×40	1.12E-02	2.98	5.95E-02	2.00	1.44E-02	3.00	6.42E-02	2.00
80×80	2.00E-03	2.49	1.18E-02	2.34	2.67E-03	2.43	1.63E-02	1.98
160×160	4.27E-04	2.23	3.12E-03	1.92	5.77E-04	2.21	6.39E-03	1.35
320×320	1.01E-04	2.08	1.29E-03	1.27	1.55E-04	1.90	1.57E-03	2.02
average	–	2.43	–	1.93	–	2.37	–	1.80
	$k=2$							
20×20	1.86E-02	–	7.37E-02	–	1.65E-02	–	7.00E-02	–
40×40	1.30E-03	3.84	1.00E-02	2.88	2.15E-03	2.94	2.37E-02	1.56
80×80	1.75E-04	2.89	1.23E-03	3.03	1.99E-04	3.43	1.90E-03	3.64
160×160	2.23E-05	2.97	2.83E-04	2.12	2.63E-05	2.92	4.75E-04	2.00
320×320	2.95E-06	2.91	3.81E-05	2.89	2.92E-06	3.17	3.90E-05	3.61
average	–	3.11	–	2.70	–	3.13	–	2.73
	$k=3$							
20×20	2.80E-03	–	1.49E-02	–	3.64E-03	–	1.48E-02	–
40×40	2.24E-04	3.64	2.32E-03	2.68	3.09E-04	3.56	2.45E-03	2.60
80×80	1.52E-05	3.89	1.82E-04	3.67	3.10E-05	3.32	2.33E-04	3.40
160×160	8.66E-07	4.13	1.09E-05	4.06	1.31E-06	4.56	1.16E-05	4.32
320×320	7.05E-08	3.62	7.60E-07	3.84	7.31E-08	4.17	7.32E-07	3.99
average	–	3.86	–	3.63	–	3.91	–	3.63

4.2 Discontinuous initial conditions

As we can see in the second numerical example in last subsection, when dealing with the solution (not in the initial time) with strong shocks, our current scheme will generate some numerical oscillations. Now we test the behavior of approximation for discontinuous initial data on random points. Here, we use the same point clouds as in Section 4.1. We first show two examples for one dimensional and two dimensional cases, respectively, and then give some conclusions.

Example 1. We first test the one-dimensional discontinuous initial condition

$$u_0(x) = \begin{cases} 1, & if\ x \leqslant \pi, \\ 0, & if\ x > \pi. \end{cases} \tag{25}$$

Table 13: 2D Burgers equation with $u(x, y, 0) = 0.5 + \sin(x + y)$ at $t = 0.25$. Error on the whole domain.

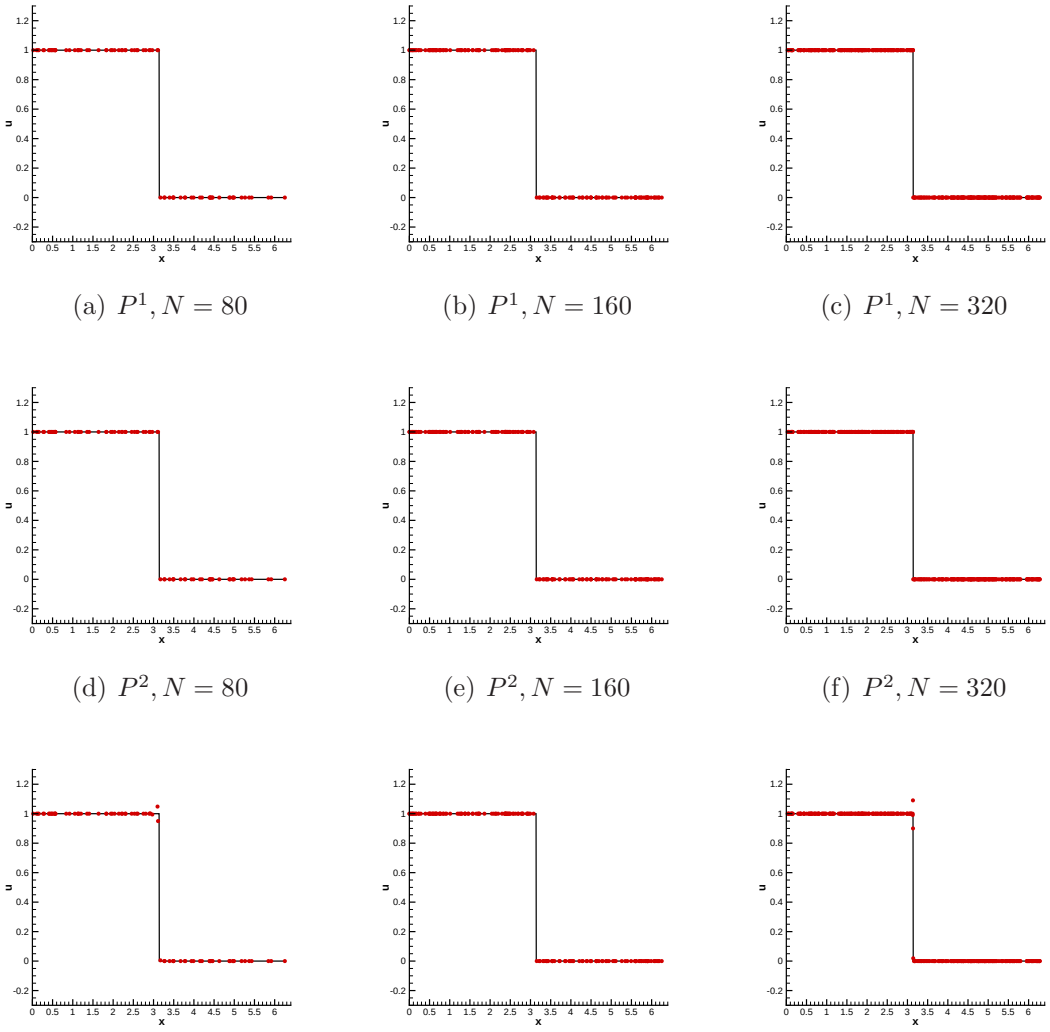
$N_x \times N_y$	uniform points				random points			
	L^1 norm	order	L^∞ norm	order	L^1 norm	order	L^∞ norm	order
	$k = 1$							
20×20	2.23E-02	–	0.33	–	5.09E-02	–	0.54	–
40×40	5.64E-03	1.98	0.11	1.56	1.27E-02	2.01	0.18	1.59
80×80	1.38E-03	2.03	3.45E-02	1.68	3.62E-03	1.81	8.43E-02	1.09
160×160	3.45E-04	2.00	8.08E-03	2.09	9.58E-04	1.92	3.72E-02	1.18
320×320	8.56E-05	2.01	2.37E-03	1.77	2.42E-04	1.99	8.14E-03	2.19
average	–	2.01	–	1.80	–	1.92	–	1.44
	$k = 2$							
20×20	9.91E-03	–	0.44	–	2.32E-02	–	0.47	–
40×40	1.45E-03	2.77	0.13	1.81	3.67E-03	2.66	0.20	1.24
80×80	1.56E-04	3.22	1.26E-02	3.31	5.05E-04	2.86	3.03E-02	2.72
160×160	2.12E-05	2.88	3.06E-03	2.05	6.92E-05	2.87	6.27E-03	2.27
320×320	2.49E-06	3.09	4.14E-04	2.89	8.83E-06	2.97	1.60E-03	1.97
average	–	3.00	–	2.54	–	2.84	–	2.14
	$k = 3$							
20×20	4.85E-03	–	0.16	–	1.38E-02	–	0.42	–
40×40	3.94E-04	3.62	2.27E-02	2.86	1.42E-03	3.28	7.83E-02	2.42
80×80	3.04E-05	3.70	2.01E-03	3.50	1.25E-04	3.50	1.14E-02	2.78
160×160	2.11E-06	3.85	1.69E-04	3.58	8.22E-06	3.93	1.47E-03	2.95
320×320	1.40E-07	3.91	1.23E-05	3.78	6.12E-07	3.75	1.44E-04	3.36
average	–	3.77	–	3.45	–	3.63	–	2.88

Given the initial values only on the random point could, we now divide all the points into groups and approximate the initial data with piecewise polynomials. Figure 12 shows the values of the initial piecewise polynomials on the given random points for different k and different N. Next, we use the approximated piecewise polynomials as initial data to solve for the nonlinear Burgers equation. Figure 13 shows the results on random points at $t = 1$.

Example 2. Consider the two-dimensional discontinuous initial condition

$$u_0(x, y) = \begin{cases} 1, & if\ x + y \leqslant 2\pi, \\ 0, & if\ x + y > 2\pi. \end{cases} \tag{26}$$

Figure 14 shows the values of the approximated piecewise polynomials on the given random points. Now we consider the two-dimensional Burgers equation with the ap-

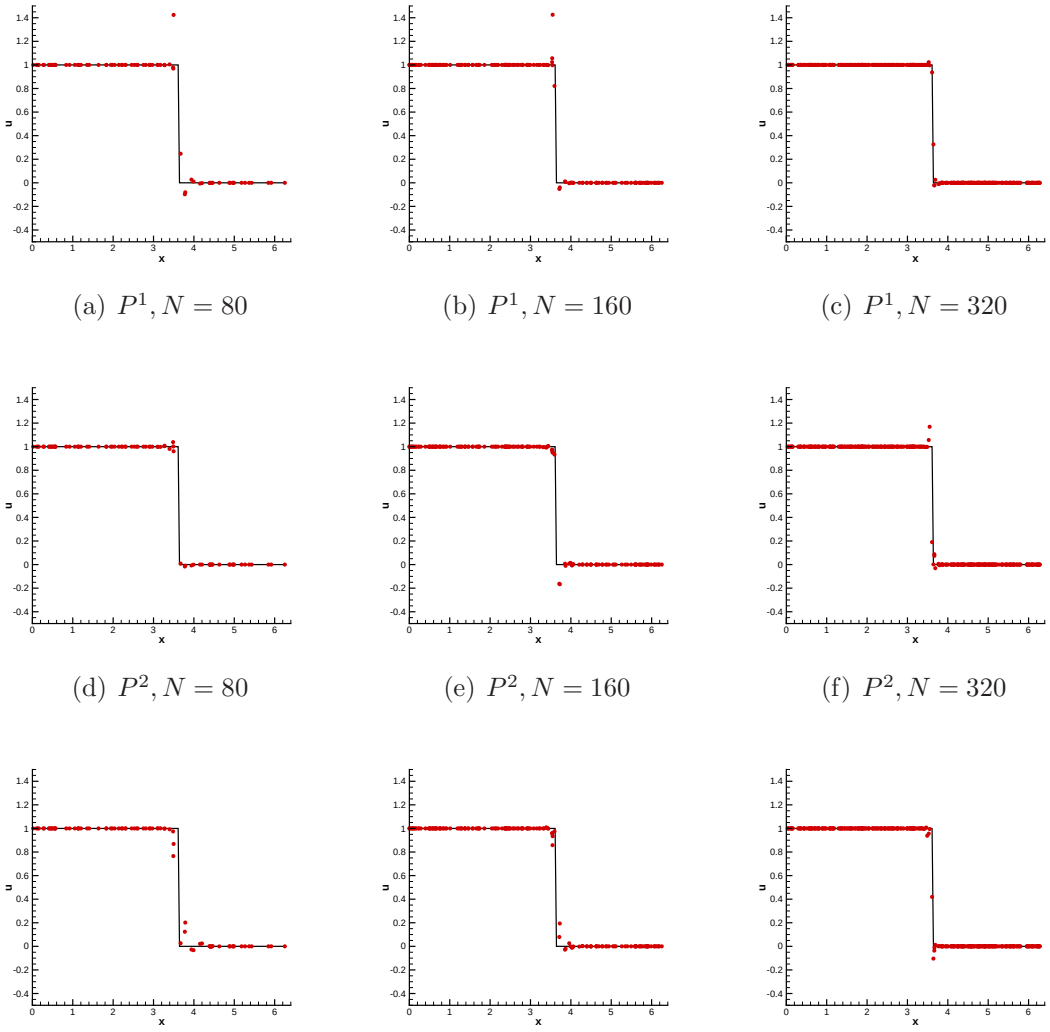


(a) $P^1, N = 80$ (b) $P^1, N = 160$ (c) $P^1, N = 320$

(d) $P^2, N = 80$ (e) $P^2, N = 160$ (f) $P^2, N = 320$

(g) $P^3, N = 80$ (h) $P^3, N = 160$ (i) $P^3, N = 320$

Figure 12: **1D approximation of discontinuous initial data on random point cloud.**



(a) $P^1, N = 80$ (b) $P^1, N = 160$ (c) $P^1, N = 320$

(d) $P^2, N = 80$ (e) $P^2, N = 160$ (f) $P^2, N = 320$

(g) $P^3, N = 80$ (h) $P^3, N = 160$ (i) $P^3, N = 320$

Figure 13: **1D Burgers with discontinuous initial function on random point cloud.** $t = 1$**.**

Table 14: 2D Burgers equation with $u(x, y, 0) = 0.5 + \sin(x + y)$ at $t = 0.25$. Error on the given points.

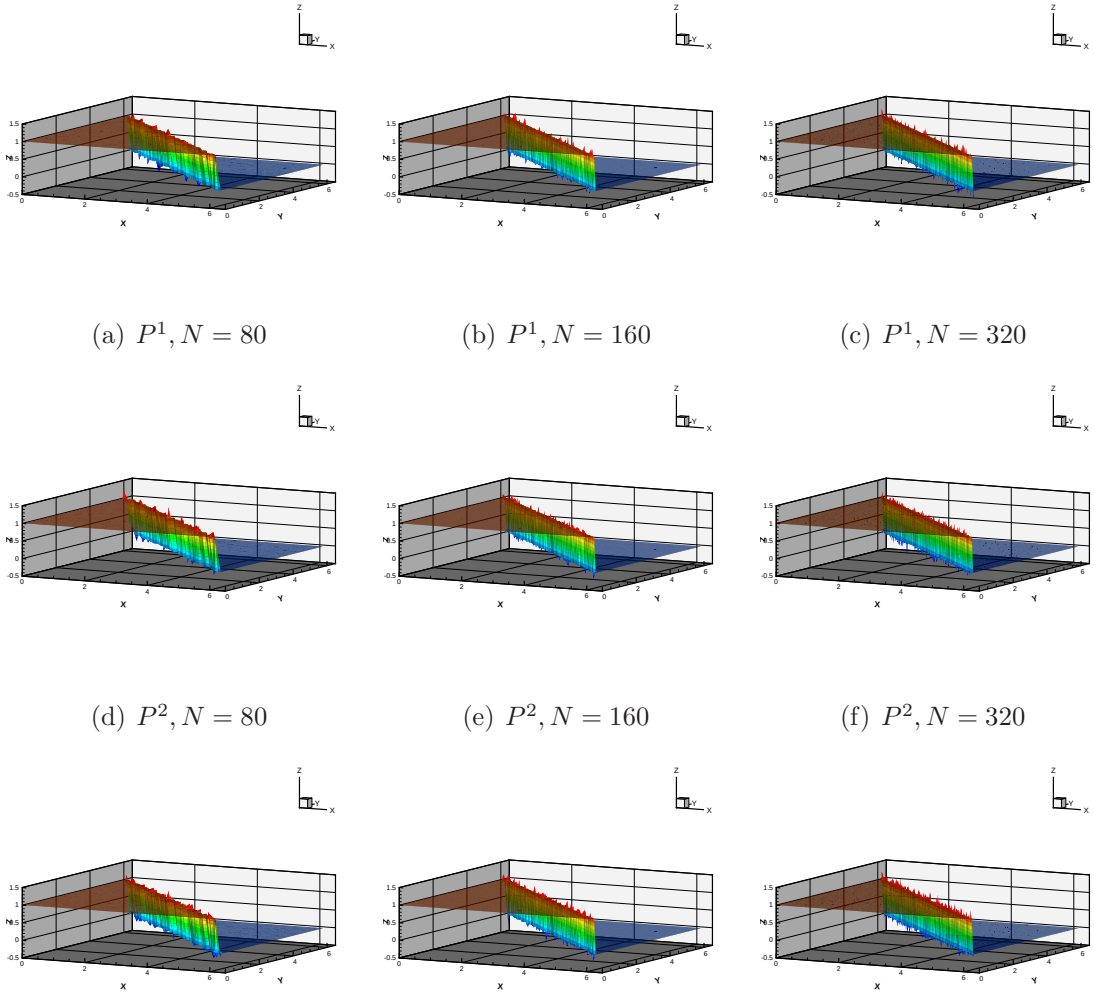
$N_x \times N_y$	uniform points				random points			
	L^1 norm	order	L^∞ norm	order	L^1 norm	order	L^∞ norm	order
	$k = 1$							
20×20	1.12E-02	–	1.02E-01	–	3.91E-02	–	2.46E-01	–
40×40	2.44E-03	2.19	2.92E-02	1.81	9.56E-03	2.03	6.91E-02	1.83
80×80	5.46E-04	2.16	7.03E-03	2.06	2.80E-03	1.77	3.90E-02	0.83
160×160	1.31E-04	2.06	2.03E-03	1.80	7.52E-04	1.89	1.61E-02	1.27
320×320	3.16E-05	2.05	5.00E-04	2.02	1.92E-04	1.97	5.53E-03	1.55
average	–	2.12	–	1.92	–	1.90	–	1.31
	$k = 2$							
20×20	7.37E-03	–	1.06E-01	–	1.88E-02	–	1.52E-01	–
40×40	1.04E-03	2.82	3.25E-02	1.70	2.89E-03	2.70	5.48E-02	1.47
80×80	1.13E-04	3.20	2.33E-03	3.80	4.21E-04	2.78	1.24E-02	2.14
160×160	1.65E-05	2.78	6.60E-04	1.82	5.73E-05	2.88	3.31E-03	1.91
320×320	1.86E-06	3.15	9.48E-05	2.80	7.44E-06	2.95	6.18E-04	2.42
average	–	2.99	–	2.59	–	2.83	–	1.99
	$k = 3$							
20×20	3.74E-03	–	3.06E-02	–	1.11E-02	–	1.00E-01	–
40×40	3.41E-04	3.45	4.16E-03	2.88	1.22E-03	3.20	2.36E-02	2.09
80×80	2.72E-05	3.65	4.07E-04	3.35	1.05E-04	3.53	3.57E-03	2.73
160×160	1.89E-06	3.84	2.99E-05	3.77	7.20E-06	3.87	3.26E-04	3.45
320×320	1.24E-07	3.93	2.50E-06	3.58	5.49E-07	3.71	4.37E-05	2.90
average	–	3.73	–	3.43	–	3.60	–	2.85

proximated piecewise polynomials as the initial data. Figure 15 shows the results at $t = 1$.

For both the one dimensional and the two dimensional cases, we can see that there are some small oscillations in the approximated piecewise polynomials near the discontinuity at the initial time, but in general we are able to recover to the true initial conditions and the approximation in the smooth region becomes more accurate as we refine the mesh.

Table 15: 2D Burgers equation. Computational costs (in s). $N_x = N_y = 160$.

order	uniform points				random points			
	Voronoi	Grouping	Initialization	DG	Voronoi	Grouping	Initialization	DG
$k = 1$	46.89	2.82E-01	3.20E-02	33.68	65.36	4.06E-01	8.60E-02	309.18
$k = 2$	–	2.89E-01	5.60E-02	29.64	–	4.02E-01	1.64E-01	210.12
$k = 3$	–	3.99E-01	1.15E-01	14.75	–	5.61E-01	2.52E-01	85.37

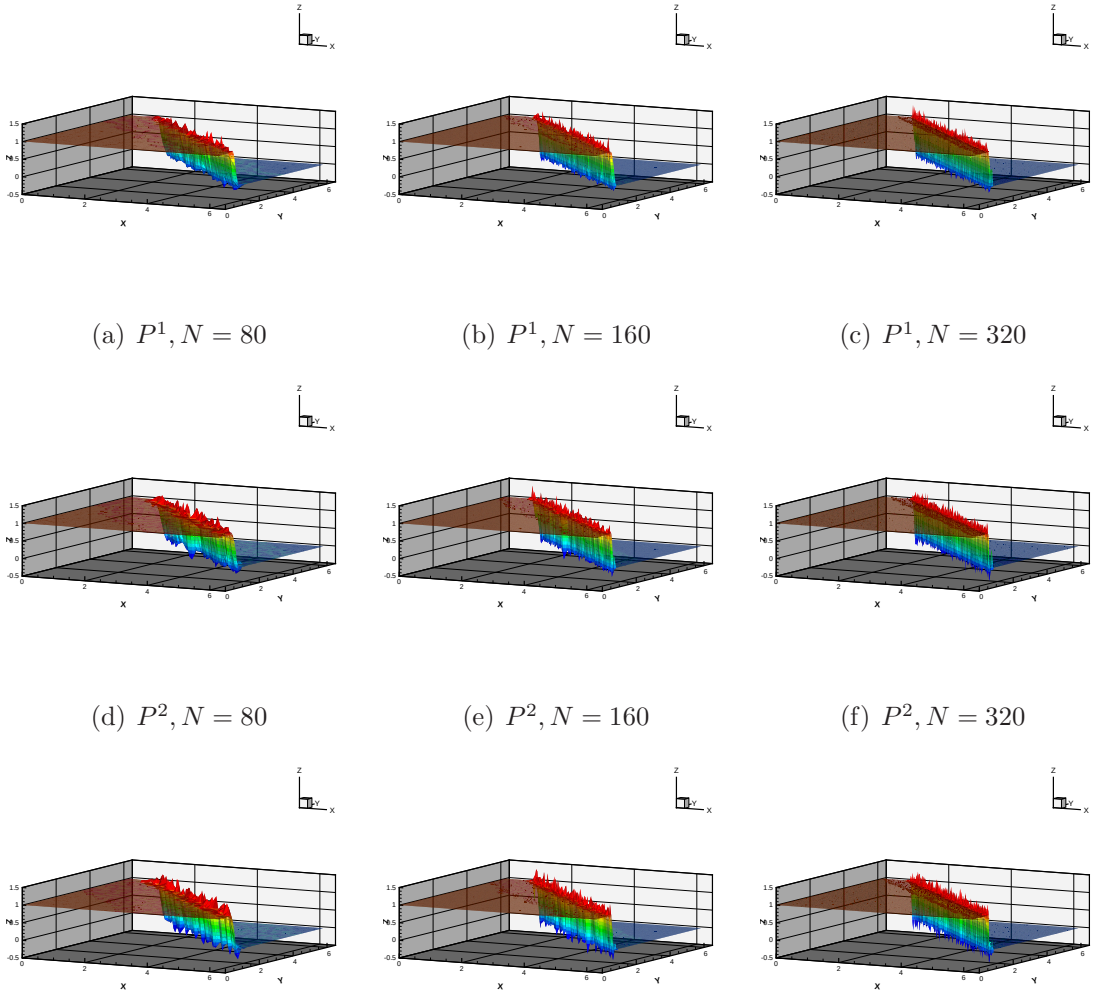


(a) $P^1, N = 80$ (b) $P^1, N = 160$ (c) $P^1, N = 320$

(d) $P^2, N = 80$ (e) $P^2, N = 160$ (f) $P^2, N = 320$

(g) $P^3, N = 80$ (h) $P^3, N = 160$ (i) $P^3, N = 320$

Figure 14: **2D approximation of discontinuous initial data on random point cloud.**



(a) $P^1, N = 80$ (b) $P^1, N = 160$ (c) $P^1, N = 320$

(d) $P^2, N = 80$ (e) $P^2, N = 160$ (f) $P^2, N = 320$

(g) $P^3, N = 80$ (h) $P^3, N = 160$ (i) $P^3, N = 320$

Figure 15: **2D Burgers with discontinuous initial function on random point cloud.** $t = 1$**.**

Table 16: 2D Euler equation with $\rho(x,y,0) = 1 + 0.2\sin(x+y)$, $u(x,y,0) = 0.7$, $v(x,y,0) = 0.3$ and $p(x,y,0) = 1$ at $t = 2\pi$. Error on the whole domain.

$N_x \times N_y$	uniform points				random points			
	L^1 norm	order	L^∞ norm	order	L^1 norm	order	L^∞ norm	order
	$k = 1$							
20×20	1.04E-02	–	3.42E-02	–	2.76E-02	–	9.90E-02	–
40×40	1.35E-03	2.95	6.79E-03	2.33	3.97E-03	2.80	2.37E-02	2.06
80×80	2.31E-04	2.54	2.32E-03	1.55	7.11E-04	2.48	8.49E-03	1.48
160×160	5.04E-05	2.20	5.14E-04	2.17	1.64E-04	2.12	3.23E-03	1.40
320×320	1.22E-05	2.05	1.55E-04	1.73	3.99E-05	2.03	1.04E-03	1.64
average	–	2.42	–	1.93	–	2.35	–	1.60
	$k = 2$							
20×20	1.62E-03	–	3.62E-02	–	5.03E-03	–	7.01E-02	–
40×40	1.47E-04	3.47	3.06E-03	3.56	5.26E-04	3.26	7.12E-03	3.30
80×80	1.41E-05	3.38	4.92E-04	2.64	5.40E-05	3.28	1.68E-03	2.08
160×160	1.74E-06	3.01	7.53E-05	2.71	6.37E-06	3.08	2.77E-04	2.60
320×320	1.95E-07	3.17	8.20E-06	3.20	7.79E-07	3.03	2.31E-05	3.58
average	–	3.24	–	2.96	–	3.17	–	2.78
	$k = 3$							
20×20	2.06E-04	–	2.19E-03	–	1.59E-03	–	1.70E-02	–
40×40	1.19E-05	4.11	1.62E-04	3.76	6.63E-05	4.59	1.48E-03	3.52
80×80	7.01E-07	4.09	1.01E-05	4.01	4.35E-06	3.93	1.19E-04	3.64
160×160	4.29E-08	4.03	6.33E-07	3.99	2.55E-07	4.09	1.37E-05	3.12
320×320	2.67E-09	4.01	3.96E-08	4.00	1.59E-08	4.01	8.48E-07	4.01
average	–	4.06	–	3.95	–	4.13	–	3.53

Also, we can see that although there are some oscillations near the discontinuity at the initial time, the scheme is stable and does not blow up. This is due to the nonlinear L^2 stability of the DG algorithm which results from the cell entropy inequalities.

Of course, when no limiters are used, there are spurious numerical oscillations around the discontinuities and these oscillations become more severe for higher order methods. This is expected, as similar oscillations also exist for DG schemes without limiters on classical triangular or rectangular meshes, even with the smooth initial conditions. Nonlinear limiters would be needed to deal with these oscillations for strong shocks, which will be studied in the future.

Table 17: 2D Euler equation with $\rho(x,y,0) = 1 + 0.2\sin(x+y)$, $u(x,y,0) = 0.7$, $v(x,y,0) = 0.3$ and $p(x,y,0) = 1$ at $t = 2\pi$. Error on the given points.

$N_x \times N_y$	uniform points				random points			
	L^1 norm	order	L^∞ norm	order	L^1 norm	order	L^∞ norm	order
	$k = 1$							
20×20	1.16E-02	–	3.09E-02	–	2.80E-02	–	8.89E-02	–
40×40	1.41E-03	3.04	4.52E-03	2.77	3.94E-03	2.83	1.56E-02	2.51
80×80	1.74E-04	3.02	6.54E-04	2.79	6.53E-04	2.59	4.93E-03	1.66
160×160	2.14E-05	3.02	1.64E-04	2.00	1.41E-04	2.21	1.58E-03	1.64
320×320	3.78E-06	2.50	2.80E-05	2.55	3.34E-05	2.08	5.33E-04	1.57
average	–	2.92	–	2.50	–	2.42	–	1.81
	$k = 2$							
20×20	1.43E-03	–	9.15E-03	–	4.42E-03	–	2.74E-02	–
40×40	1.23E-04	3.54	9.38E-04	3.29	4.53E-04	3.29	4.11E-03	2.74
80×80	1.15E-05	3.41	1.25E-04	2.91	4.71E-05	3.27	8.26E-04	2.31
160×160	1.45E-06	2.99	1.58E-05	2.98	5.47E-06	3.11	9.23E-05	3.16
320×320	1.60E-07	3.18	2.02E-06	2.97	6.71E-07	3.03	1.05E-05	3.13
average	–	3.27	–	3.02	–	3.17	–	2.82
	$k = 3$							
20×20	1.85E-04	–	4.75E-04	–	1.42E-03	–	6.31E-03	–
40×40	1.02E-05	4.18	3.08E-05	3.95	5.95E-05	4.58	6.69E-04	3.24
80×80	5.92E-07	4.11	1.99E-06	3.95	3.90E-06	3.93	4.06E-05	4.04
160×160	3.61E-08	4.03	1.25E-07	3.99	2.29E-07	4.09	9.67E-06	2.07
320×320	2.24E-09	4.01	7.87E-09	3.99	1.41E-08	4.02	2.19E-07	5.46
average	–	4.08	–	3.97	–	4.13	–	3.57

5 Concluding remarks

In this paper, we aim to solve one and two dimensional time-dependent hyperbolic conservation laws, with initial values only given at an arbitrarily distributed point cloud. With the given point cloud, we first divide the computational domain into a Voronoi diagram. Each region within the Voronoi diagram contains one point in the give point cloud and consists of all locations in the computational domain closer to that point than to any other. Then we group these regions into cells. Each cell is a polygon and consists of several neighboring Voronoi regions. By controlling the condition number of the matrix used in the interpolation procedure, we carefully select points in each cell and hence are able to interpolate or fit the discrete initial values with piecewise polynomials. By adapting the traditional DG method on the constructed mesh, we

Table 18: 2D Euler system. The smooth vortex evolution problem at $t = 0.2$. Error on the whole domain.

$N_x \times N_y$	uniform points				random points			
	L^1 norm	order	L^∞ norm	order	L^1 norm	order	L^∞ norm	order
	$k = 1$							
20×20	3.27E-03	–	0.15	–	5.12E-03	–	0.18	–
40×40	1.09E-03	1.58	7.30E-02	1.02	1.45E-03	1.82	8.28E-02	1.12
80×80	2.94E-04	1.90	2.39E-02	1.61	4.71E-04	1.62	4.34E-02	0.93
160×160	7.29E-05	2.01	6.70E-03	1.83	1.41E-04	1.74	1.23E-02	1.82
320×320	1.80E-05	2.02	1.74E-03	1.95	3.68E-05	1.94	5.78E-03	1.09
average	–	1.89	–	1.63	–	1.76	–	1.27
	$k = 2$							
20×20	2.33E-03	–	9.41E-02	–	4.36E-03	–	0.20	–
40×40	4.20E-04	2.47	3.79E-02	1.31	8.53E-04	2.35	6.95E-02	1.49
80×80	6.66E-05	2.66	7.10E-03	2.42	1.13E-04	2.91	7.65E-03	3.18
160×160	8.82E-06	2.92	9.64E-04	2.88	1.87E-05	2.60	2.85E-03	1.42
320×320	1.53E-06	2.53	1.87E-04	2.36	2.72E-06	2.78	3.70E-04	2.95
average	–	2.67	–	2.32	–	2.68	–	2.27
	$k = 3$							
20×20	3.64E-03	–	0.16	–	2.92E-03	–	8.62E-02	–
40×40	3.32E-04	3.46	2.89E-02	2.43	5.84E-04	2.32	2.99E-02	1.53
80×80	2.45E-05	3.76	4.62E-03	2.65	4.98E-05	3.55	4.54E-03	2.72
160×160	1.82E-06	3.75	3.02E-04	3.94	4.48E-06	3.47	7.23E-04	2.65
320×320	1.18E-07	3.94	2.18E-05	3.79	2.83E-07	3.99	6.90E-05	3.39
average	–	3.73	–	3.22	–	3.37	–	2.59

obtain a conservative, stable and high order method. Numerical examples for both one and two dimensional scalar equations and Euler systems of compressible gas dynamics are provided to illustrate the good behavior of our mesh generation algorithm and the numerical scheme.

As we can see in the numerical example, when dealing with solutions containing strong shocks, our current scheme will generate spurious numerical oscillations, just as DG schemes without limiters on regular triangular or rectangular meshes will do. In our future work, we will develop limiters for our polygonal mesh to eliminate the oscillations near discontinuities as well as to maintain uniform high order accuracy in smooth regions, such as the WENO limiters and positivity-preserving limiters used for the DG method [54, 55, 53] and for the CPR method [18, 19].

Table 19: 2D Euler system. The smooth vortex evolution problem at $t = 0.2$. Error on the given points.

$N_x \times N_y$	uniform points				random points			
	L^1 norm	order	L^∞ norm	order	L^1 norm	order	L^∞ norm	order
	$k = 1$							
20×20	2.29E-03	–	9.11E-02	–	4.18E-03	–	1.13E-01	–
40×40	7.69E-04	1.57	2.36E-02	1.95	1.09E-03	1.93	6.15E-02	0.87
80×80	1.95E-04	1.98	6.55E-03	1.85	3.82E-04	1.52	1.66E-02	1.88
160×160	4.69E-05	2.06	1.72E-03	1.93	1.20E-04	1.67	8.28E-03	1.01
320×320	1.13E-05	2.06	4.12E-04	2.06	3.16E-05	1.92	2.51E-03	1.72
average	–	1.94	–	1.94	–	1.73	–	1.39
	$k = 2$							
20×20	1.67E-03	–	5.58E-02	–	3.20E-03	–	7.55E-02	–
40×40	3.13E-04	2.42	1.27E-02	2.14	6.66E-04	2.27	2.43E-02	1.63
80×80	4.85E-05	2.69	2.02E-03	2.65	9.52E-05	2.81	3.33E-03	2.87
160×160	6.69E-06	2.86	3.52E-04	2.52	1.65E-05	2.53	1.14E-03	1.55
320×320	1.14E-06	2.55	8.17E-05	2.11	2.47E-06	2.73	2.45E-04	2.22
average	–	2.66	–	2.40	–	2.60	–	2.09
	$k = 3$							
20×20	2.39E-03	–	4.35E-02	–	2.77E-03	–	6.97E-02	–
40×40	2.59E-04	3.21	9.22E-03	2.24	5.01E-04	2.47	1.41E-02	2.30
80×80	1.98E-05	3.71	1.05E-03	3.14	4.42E-05	3.50	2.70E-03	2.39
160×160	1.42E-06	3.80	1.11E-04	3.24	4.12E-06	3.42	6.39E-04	2.08
320×320	9.67E-08	3.87	7.57E-06	3.87	2.66E-07	3.96	5.07E-05	3.66
average	–	3.67	–	3.14	–	3.36	–	2.53

In this paper, we only present the method for solving time-dependent conservation laws on arbitrarily distributed point clouds, but our method can also be applied to the convection-diffusion problems, which will be studied in the future.

References

[1] H.L. Atkins and C.-W. Shu, *Quadrature-free implementation of discontinuous Galerkin method for hyperbolic equations*, AIAA Journal, 36 (1998), 775–782.

[2] I. Babuška, U. Banerjee and J.E. Osborn, *Meshless and generalized finite element methods: a survey of some major results*, in *Meshfree Methods for Partial Differential Equations*, Springer, Berlin and Heidelberg, 2003, 1–20.

[3] I. Babuška, U. Banerjee and J.E. Osborn, *Survey of meshless and generalized finite element methods: a unified approach*, Acta Numerica, 12 (2003), 1–125.

[4] I. Babuška and J.M. Melenk, *The partition of unity finite element method*, Technical Note BN-1185, Univ. of Maryland, 1995.

[5] L. Beirão da Veiga, F. Brezzi, A. Cangiani, G. Manzini, L.D. Marini and A. Russo, *Basic principles of virtual element methods*, Mathematical Models and Methods in Applied Sciences (M^3AS), 23 (2013), 199–214.

[6] W. Benz, *Smooth particle hydrodynamics: a review*, in *The Numerical Modeling of Non-linear Stellar Pulsation*, Springer, Netherlands, 1990, 269–288.

[7] Å. Björck, *Least Squares Methods*, Handbook of Numerical Analysis, 1 (1990), 465–652.

[8] P. Breitkopf, G. Touzot and P. Villon, *Consistency approach and diffuse derivation in element free methods based on moving least squares approximation*, Computer Assisted Mechanics and Engineering Sciences, 5 (1998), 479–501.

[9] B. Cockburn, *An introduction to the discontinuous Galerkin method for convection-dominated problems*, in *Advanced Numerical Approximation of Nonlinear Hyperbolic Equations*, Springer, Berlin and Heidelberg, 1998, 151–268.

[10] B. Cockburn, S. Hou and C.-W. Shu, *The Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws IV: the multidimensional case*, Mathematics of Computation, 54 (1990), 545–581.

[11] B. Cockburn, S.-Y. Lin and C.-W. Shu, *TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws III: one dimensional systems*, Journal of Computational Physics, 84 (1989), 90–113.

[12] B. Cockburn and C.-W. Shu, *TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws II: general framework*, Mathematics of Computation, 52 (1989), 411–435.

[13] B. Cockburn and C.-W. Shu, *The Runge-Kutta local projection P1-discontinuous-Galerkin finite element method for scalar conservation laws*, Mathematical Modelling and Numerical Analysis (M2AN), 25 (1991), 337–361.

[14] B. Cockburn and C.-W. Shu, *The Runge-Kutta discontinuous Galerkin method for conservation laws V: multidimensional systems*, Journal of Computational Physics, 141 (1998), 199–224.

[15] M. Dahleh, M.A. Dahleh and G. Verghese, *Lectures on dynamic systems and control*, A+ A, 4 (2004), 1–100.

[16] V. Dolejsi, M. Feistauer, and J. Hozman, *Analysis of semi-implicit DGFEM for nonlinear convection-diffusion problems on nonconforming meshes*, Computer Methods in Applied Mechanics and Engineering, 196 (2007), 2813–2827.

[17] V. Dolejsi, M. Feistauer, V. Sobotikova, *Analysis of the discontinuous Galerkin method for nonlinear convection-diffusion problems*, Computer Methods in Applied Mechanics and Engineering, 194 (2005), 2709–2733.

[18] J. Du, C.-W. Shu and M. Zhang, *A simple weighted essentially non-oscillatory limiter for the correction procedure via reconstruction (CPR) framework*, Applied Numerical Mathematics, 95 (2015), 173–198.

[19] J. Du, C.-W. Shu and M. Zhang, *A simple weighted essentially non-oscillatory limiter for the correction procedure via reconstruction (CPR) framework on unstructured meshes*, Applied Numerical Mathematics, 90 (2015), 146–167.

[20] Q. Du, V. Faber and M. Gunzburger, *Centroidal Voronoi tessellations: applications and algorithms*, SIAM Review, 41 (1999), 637–676.

[21] S. Fortune, *A sweepline algorithm for Voronoi diagrams*, Algorithmica, 2 (1987), 153–174.

[22] R.A. Gingold and J.J. Monaghan, *Smoothed particle hydrodynamics: theory and application to non-spherical stars*, Monthly Notices of the Royal Astronomical Society, 181 (1977), 375–389.

[23] S. Gottlieb and C.-W. Shu, *Total variation diminishing Runge-Kutta schemes*, Mathematics of Computation, 67 (1998), 73–85.

[24] S. Gottlieb, C.-W. Shu and E. Tadmor, *Strong stability preserving high order time discretization methods*, SIAM Review, 43 (2001), 89–112.

[25] Y. Gu and G.-R. Liu, *Meshless techniques for convection dominated problems*, Computational Mechanics, 38 (2006), 171–182.

[26] J. Hesthaven and T. Warburton, *Nodal Discontinuous Galerkin Methods*, Springer, New York, 2008.

[27] S. Hou and X.-D. Liu, *Solutions of multi-dimensional hyperbolic systems of conservation laws by square entropy condition satisfying discontinuous Galerkin method*, Journal of Scientific Computing, 31 (2007), 127–151.

[28] H.T. Huynh, *A flux reconstruction approach to high-order schemes including discontinuous Galerkin methods*, AIAA Paper 2007–4079.

[29] G.-S. Jiang and C.-W. Shu, *On cell entropy inequality for discontinuous Galerkin methods*, Mathematics of Computation, 62 (1994), 531–538.

[30] C. Johnson and J. Pitkäranta, *An analysis of the discontinuous Galerkin method for a scalar hyperbolic equation*, Mathematics of Computation, 46 (1986), 1–26.

[31] P. Lancaster and K. Salkauskas, *Surface generated by moving least square methods*, Mathematics of Computation, 37 (1981), 141–158.

[32] S. Li and W.K. Liu, *Meshfree and particle methods and their applications*. Applied Mechanics Reviews, 55 (2002), 1–34.

[33] W.K. Liu, Y. Chen, S. Jun, J.S. Chen, T. Belytschko, R.A. Uras and C.T. Chang, *Overview and applications of the reproducing kernel particle methods*, Archives of Computational Methods in Engineering, 3 (1996), 3–80.

[34] L.B. Lucy, *A numerical approach to the testing of the fission hypothesis*, The Astronomical Journal, 82 (1977), 1013–1024.

[35] J.M. Melenk and I. Babuška, *The partition of unity finite element method: basic theory and applications*, Computer Methods in Applied Mechanics and Engineering, 139 (1996), 289–314.

[36] J.J. Monaghan, *Particle methods for hydrodynamics*, Computer Physics Reports, 3 (1985), 71–124.

[37] B. Nayroles, G. Touzot and P. Villon, *Generalizing the finite element method: diffuse approximation and diffuse elements*, Computational Mechanics, 10 (1992), 307–318.

[38] A. Okabe, B. Boots, and K. Sugihara, *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*, Wiley, Chichester, UK, 1992.

[39] S. O'Sullivan, `http://www.skynet.ie/~sos/mapviewer/voronoi.php`.

[40] W.H. Reed and T.R. Hill, *Triangular mesh methods for the neutron transport equation*, Tech. Report LA-UR-73-479, Los Alamos Scientific Laboratory, 1973.

[41] G.R. Richter, *An optimal-order error estimate for the discontinuous Galerkin method*, Mathematics of Computation, 50 (1988), 75–88.

[42] D. Shepard, *A two-dimensional interpolation function for irregularly spaced points*, Proceedings of the 1968 23rd ACM National Conference, ACM, 1968, 517–524.

[43] C. Shu, H. Ding and K. S. Yeo, *Local radial basis function-based differential quadrature method and its application to solve two-dimensional incompressible Navier–Stokes equations* Computer Methods in Applied Mechanics and Engineering, 192 (2003), 941–954.

[44] C.-W. Shu, *Total-Variation-Diminishing time discretizations*, Scientific and Statistical Computing, 9 (1988), 1073–1084.

[45] C.-W. Shu, *Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws*, in *Advanced Numerical Approximation of Nonlinear Hyperbolic Equations*, Lecture Notes in Mathematics, volume 1697, Springer, Berlin, 1998, 325–432.

[46] C.-W. Shu, *Discontinuous Galerkin methods: general approach and stability*, in *Numerical Solutions of Partial Differential Equations*, Advanced Courses in Mathematics CRM Barcelona, Birkhäuser, Basel, 2009, 149-201.

[47] C.-W. Shu and S. Osher, *Efficient implementation of essentially non-oscillatory shock-capturing schemes*, Journal of Computational Physics, 77 (1988), 439–471.

[48] Siraj-ul-Islam, R. Vertnik and B. Šarler, *Local radial basis function collocation method along with explicit time stepping for hyperbolic partial differential equations*, Applied Numerical Mathematics, 67 (2013), 136–151.

[49] Z.J. Wang and H. Gao, *A unifying lifting collocation penalty formulation including the discontinuous Galerkin, spectral volume/difference methods for conservation laws on mixed grids*, Journal of Computational Physics, 228 (2009), 8161–8186.

[50] Q. Zhang and C.-W. Shu, *Error estimates to smooth solutions of Runge-Kutta discontinuous Galerkin methods for scalar conservation laws*, SIAM Journal on Numerical Analysis, 42 (2004), 641–666.

[51] Q. Zhang and C.-W. Shu, *Error estimates to smooth solutions of Runge-Kutta discontinuous Galerkin method for symmetrizable systems of conservation laws*, SIAM Journal on Numerical Analysis, 44 (2006), 1703–1720.

[52] Q. Zhang and C.-W. Shu, *Stability analysis and a priori error estimates of the third order explicit Runge-Kutta discontinuous Galerkin method for scalar conservation laws*, SIAM Journal on Numerical Analysis, 48 (2010), 1038–1063.

[53] X. Zhang and C.-W. Shu, *On positivity-preserving high order discontinuous Galerkin schemes for compressible Euler equations on rectangular meshes*, Journal of Computational Physics, 229 (2010), 8918–8934.

[54] X. Zhong and C.-W. Shu, *A simple weighted essentially nonoscillatory limiter for RungeCKutta discontinuous Galerkin methods*, Journal of Computational Physics, 232 (2013), 397–415.

[55] J. Zhu, X. Zhong, C.-W. Shu and J.-X. Qiu, *RungeCKutta discontinuous Galerkin method using a new type of WENO limiters on unstructured meshes*, Journal of Computational Physics, 248 (2013), 200–220.