VertexShuffle-Based Spherical Super-Resolution for 360-Degree Videos

NA LI, Rutgers University, USA
YAO LIU, Rutgers University, USA

360-degree video is an emerging form of media that encodes information about all directions surrounding a camera, offering an immersive experience to the users. Unlike traditional 2D videos, visual information in 360-degree videos can be naturally represented as pixels on a sphere. Inspired by state-of-the-art deep-learning-based 2D image super-resolution models and spherical CNNs, in this paper, we design a novel spherical super-resolution (SSR) approach for 360-degree videos. To support viewport-adaptive and bandwidth-efficient transmission/streaming of 360-degree video data and save computation, we propose the Focused Icosahedral Mesh to represent a small area on the sphere. We further construct matrices to rotate spherical content over the entire sphere to the focused mesh area, allowing us to use the focused mesh to represent any area on the sphere. Motivated by the PixelShuffle operation for 2D super-resolution, we also propose a novel VertexShuffle operation on the mesh and an improved version VertexShuffle_V2. We compare our SSR approach with state-of-the-art 2D super-resolution models and show that SSR has the potential to achieve significant benefits when applied to spherical signals.

CCS Concepts: • **Information systems** → **Multimedia streaming**; • **Computing methodologies** → **Image processing**.

Additional Key Words and Phrases: VertexShuffle, 360-degree videos, Focused Icosahedral Mesh

ACM Reference Format:
Na Li and Yao Liu. 2024. VertexShuffle-Based Spherical Super-Resolution for 360-Degree Videos. *ACM Trans. Multimedia Comput. Commun. Appl.* 1, 1, Article 1 (January 2024), 17 pages. https://doi.org/10.1145/3649315

1 INTRODUCTION

360-degree image/video, also known as spherical image/video, is an emerging format of media that captures views from all directions surrounding the camera. Unlike traditional 2D image/video that limits the user's view to wherever the camera is facing during capturing, a 360-degree image/video allows the viewer to freely navigate a full omnidirectional scene around the camera position.

Despite its substantial promise of immersiveness, the utility of streaming/transmission of 360-degree video is limited by the huge bandwidths required by most implementations. For example, when watching a 360-degree video, users can only watch a small portion of the full omnidirectional view. That is, while the 360-degree video encodes frames that cover the full $360^\circ \times 180^\circ$ field-of-view (FoV), the user may only observe a "view" (e.g., $100^\circ \times 100^\circ$ FoV) of the omnidirectional frame at a time. If the omnidirectional frame is projected to the 2D frame using the equirectangular projection [4], then only roughly 15% of the pixels on the frame is viewed (assuming $100^\circ \times 100^\circ$ FoV and the center of the viewport is aligned with the equator of the equirectangular projection). The rest 85% pixels are not viewed, resulting in significant bandwidth waste when streaming 360-degree videos. To improve the bandwidth efficiency, a number of spatial adaptation approaches have been proposed, e.g., [16, 24, 40, 41, 49, 54]. A core difficulty with such approaches involves predicting

Authors' addresses: Na Li, Rutgers University, Piscataway, NJ, USA, na.li@rutgers.edu; Yao Liu, Rutgers University, Piscataway, NJ, USA, yao.liu@rutgers.edu.

which portions of the 360-degree frame will be viewed by the user. If accurate predictions can be made of the user's behavior (a difficult task), then only the portions of the 360-degree frame to be viewed need to be transmitted. When accurate prediction may not be possible, a "soft" version of this family of approaches involves transmitting spatial portions of the 360-degree video at a level of quality in proportion to the probability that they will be viewed [59]. These approaches, however, can still suffer from difficulties, as there can be significant delays between the time that view prediction is made and the time that spatial portions of the sphere are actually viewed, resulting in inaccurate predictions [7].

A recent approach toward mitigating these types of mis-predictions involves applying super-resolution (SR) to low-resolution video segments already present in the playback buffer, e.g., [11, 17, 52]. The super-resolution task reconstructs a high-resolution image/frame from low-resolution input [27]. To date, many approaches for super-resolution over standard 2D images via deep convolutional network have been proposed, e.g., [10, 18–20, 35, 36, 50, 57, 58]. However, a problem with the current practice of 360-degree videos is the distortions caused by spherical-to-2D projections. The omnidirectional views captured by 360-degree cameras are most naturally represented as uniformly dense pixels over the surface of a sphere. When spherical pixels are projected to planar surfaces, distortions are introduced. For example, the equirectangular projection [4] is a widely used spherical projection for representing 360-degree data. However, significant distortions occur around the north and south pole areas in the projection. Such distortions can reduce the efficiency of convolutional neural network (CNN) operations by adding "over-represented" pixels, Further, training a CNN directly on the distorted representation could cause CNN models to learn characteristics of the planar distortion rather than relevant details of the high resolution representation [6, 42].

Recent works on spherical CNNs [13, 28] perform convolutional operations directly on spherical signals to avoid the distortion issue. These works show that it is possible to analyze spherical signals directly without 2D projections. Furthermore, extensive experiments were conducted in these works to show the efficiency of their proposed spherical CNNs.

In this paper, inspired by recent advances in spherical CNNs [13, 28] and state-of-the-art 2D super-resolution methods [18–20, 50, 58], we propose a spherical super-resolution (SSR) approach that operates on a direct, mesh representation of spherical pixels of 360-degree videos. First, we propose an efficient mesh representation, the Focused Icosahedral Mesh, This representation both makes our SR approach compatible with 360-degree spatial adaptation and improves memory efficiency of the training and prediction steps of model operation compared to Full Icosahedral Mesh. Second, motivated by the 2D PixelShuffle operation [47], we propose a novel VertexShuffle operation. The VertexShuffle operation significantly increases both the visual quality metric (peak signal-to-noise ratio (PSNR)) and improves inference time over comparable transposed convolution operations.

An earlier version of this work was published at NOSSDAV'22 [33]. In this manuscript, we create a new and improved operation, VertexShuffle_V2, for performing spherical super-resolution. Evaluation results show that with the new VertexShuffle_V2 operation, our spherical super-resolution approach can achieve 32.31 dB PSNR on average when super-resoluting 16x vertices on the mesh created from 360-degree video inputs. We also conduct new performance evaluations using two different training strategies: a "per segment" training strategy that trains a model for each video segment, and a "full video" training strategy that trains a single model for one full video. We also evaluated the performance our SSR approaches with baseline approaches in two different upscaling factors, $\times 4$ and $\times 8$.

In summary, this paper makes the following main contributions:[1]

[1]The repository for this project is available at https://github.com/symmru/SSR.

- We create a Focused Icosahedral Mesh representation of the sphere to efficiently represent spherical pixels in 360-degree videos. This representation not only saves computational resources but also improves memory and storage efficiencies.
- We create a novel VertexShuffle operation, inspired by the 2D PixelShuffle [47] operation, and an improved version VertexShuffle_V2. The vertex operation substantially increases both the visual quality metric (peak signal-to-noise ratio (PSNR)) and inference time over comparable transposed convolution operations.
- Results show that our proposed SSR model achieves great super-resolution performance on 360-degree video frame inputs, achieving 32.31 dB PSNR on average when super-resoluting 16x vertices (×4 upscaling factor) on the mesh and 30.11 dB PSNR on average with an even higher ×8 upscaling factor (64x vertices).

2 RELATED WORK

2.1 360-degree videos

With the advancement of AR/VR technologies, 360-degree videos have become increasingly popular among content creators and users. Compared to traditional 2D videos that have a limited and fixed field of view (FoV), 360-degree videos encodes information about all directions surrounding a camera, allowing users to freely choose the viewport in any directions, providing immersive viewing experiences. Visual information in 360-degree videos can be naturally represented as pixels on a sphere. Today, existing systems leverage 2D video codecs (e.g., H.264 and HEVC) for 360-degree video encoding by first projecting spherical pixels to 2D planar frames. Different sphere-to-2D projections exist, such as the equirectangluar projection, standard cubic projection, equi-angular cubic projection [2], and offset projections [1, 3, 59, 60].

Despite its potential for delivering more-immersive viewing experiences, current 360-degree video implementations require bandwidths that are too high to deliver satisfying experiences for many users. Numerous approaches have been proposed for improving 360-degree bandwidth efficiency. These approaches have both attempted to improve the efficiency of how the 360-degree view is represented during transmission, e.g., via a tiling approach [16, 23, 24, 40, 41, 43, 44, 49, 54, 55] as well as improving a system's ability to avoid delivering unviewed pixels [59, 61].

2.2 Spherical convolutional neural networks

Spherical CNN has been studied by the computer vision community recently as a number of real-world applications require processing signals in the spherical domain. These applications include self-driving cars, 360-degree videos, omnidirectional RGBD images, and climate science [28].

Recent works such as Cohen et al. [13] gave theoretical support of spherical CNNs for rotation-invariant learning problems, which is important for problems where orientation is crucial to the model performance. They first introduced the concepts of S^2 and $SO(3)$. S^2 can be defined as the set of points on a unit sphere, and $SO(3)$ is the rotation group in Euclidean three dimensional space. They replaced planar correlation with spherical correlation, which can be understood as the value of output feature map evaluated at rotation $R \in SO(3)$ computed as an inner product between the input feature map and a filter, rotated by R. Furthermore, they implemented the generalized Fourier transform for S^2 and $SO(3)$. Later, Cohen et al. [14] introduced a theory that is equi-variant to symmetry transformations on manifolds. They further prompt a gauge equi-variant CNN for signals on the icosahedron using the icosahedral CNN, which implements gauge equi-variant convolution using a single conv2d call, making it a highly scalable and practical alternative to spherical CNNs.

Jiang et al. presented spherical CNNs on unstructured grids (UGSCNN) [28] using parameterized differential operators for spherical signals. It introduces a basic convolution operation, called

MeshConv, that can be applied on meshes directly. It achieves significantly better performance and parameter efficiency compared to state-of-the-art network architectures for 3D classification tasks since it does not require large amounts of geodetic computations and interpolations. Zhang et al. [56] proposed to perform semantic segmentation on omnidirectional images by designing an orientation-aware CNN framework for the icosahedron mesh. They introduced fast interpolation of kernel convolutions and presented weight transfer from learned through classical CNNs to perspective data. Eder et al. [21] proposed a spherical image representation that mitigates spherical distortion by rendering a set of oriented, low-distortion images tangent to icosahedron faces. They also presented the utilities of their approaches by applying standard CNN to the spherical data.

While these existing works demonstrate their effectiveness in classification and segmentation tasks, the super-resolution task was not considered. In this work, we find it possible to apply their work to the super-resolution task. Our work is based on the MeshConv operation proposed by Jiang et al. [28] since it achieves better performance and parameter efficiency than other spherical convolutional networks. We also conduct experiments to show significant improvements over the baseline spherical super-resolution model that uses the simple MeshConv Transpose operation [28].

2.3 Super-resolution

Research in super-resolution has advanced rapidly with the success of recent deep learning models. The SRCNN [18, 19] model was the first to apply CNNs to super-resolution. FSRCNN [20] was an evolution of SRCNN. It operated directly on a low-resolution input image and applied a deconvolution layer to generate the high-resolution output. VDSR [29] was the first to apply residual layers [25] to the SR task, allowing for deeper SR networks. DRCN [30] introduced recursive learning in a very deep network for parameter sharing. Shi et al. [47] proposed "PixelShuffle", a method for mapping values at low-resolution positions directly to positions in a higher-resolution image more efficiently than the deconvolution operation. SRResNet [32] introduced a modified residual layer tailored for the SR application. An enhanced deep super-resolution network (EDSR) [37] further modified the SR-specific residual layer from SSResNet and introduced a multi-task objective in multi-scale deep super-resolution (MDSR) to support different upscaling factors in a single model. SRGAN [32] applied a Generative Adversarial Network (GAN) [22] to SR, allowing better resolution of high-frequency details. With the recent advances of the Transformer model [51], a number of works [12, 36, 62] also adopt the Transformer model in super-resolution.

The above-mentioned works all focus on 2D planar images. However, due to the distortions introduced in the sphere-to-2D projections, existing solutions may not be ideal for 360-degree image super-resolution [6, 52]. In this work, we propose to a spherical super-resolution model that directly operates on spherical signals so that we can avoid the distortion issue.

Focusing on optimizing 360-degree video streaming, Chen et al. [11] proposed to apply existing 2D super-resolution to 360-degree videos, training one model for each video. To avoid wasted computation due to limited FoV, super-resolution is only applied to selected tiles. Dasari et al. [17] proposed PARSEC that trains micro-models for video segments instead of the full video. Considering the cloud-edge-client streaming scenario for 360-degree videos, Sarkar et al. proposed L3BOU [46] that leverages the edge servers for low latency and low bandwidth streaming. To further reduce the bandwidth requirement of 360-degree video streaming, Luo et al. proposed Masked360 [39] that only needs to transmit masked frames instead of complete frames. Received frames are then restored via an autoencoder and upsampled using super-resolution. To handle the distortion caused in equirectangular projection of 360-degree images, Nishiyama et al. [42] proposed to use a distortion map that stores per-pixel scaling factors as an additional input to the super-resolution model, and S3PO [6] uses a spatial attention mechanism [53] to address the spatial distortion caused

by the equirectangular projection. Unlike their works, we focus on designing a novel spherical super-resolution approach for 360-degree images/videos.

3 METHODOLOGY

In this section, we first introduce Focused icosahedral mesh for representing a small area of the sphere. We then illustrate the novel VertexShuffle operation after a brief discussion of the MeshConv operation proposed by Jiang et al. [28]. Finally, we describe our model architecture and loss function used in model training.

3.1 Focused Icosahedral Mesh

The icosahedral spherical mesh [8] is a discretization of the spherical surface. It starts with a unit icosahedron (i.e., an icosahedron with all 12 vertices re-projected to the unit sphere). The icosahedral spherical mesh can then be obtained by first selecting midpoints of all edges, progressively sub-dividing each face of the unit icosahedron into four equal triangles by connecting 3 midpoints of the face, and re-projecting the midpoints to the unit sphere.

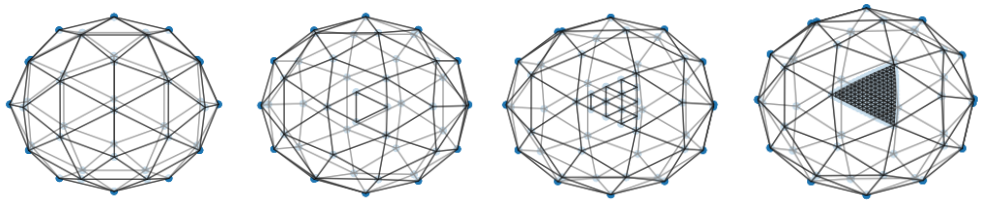
Each time every face on the icosahedral spherical mesh is refined, the mesh size increases by ≈4x. When the full icosahedral spherical mesh is refined to a granularity that can include approximately all pixels from a planar representation of a 360-degree video frame, operations would require a significant amount of computation.

Furthermore, operations on the full mesh cannot easily support operations on sub-areas of the spherical surface. Performing super-resolution on "sub-areas" of the spherical surface can be beneficial for real-world 360-degree applications such as 360-degree video streaming. This is because human eyes as well as their viewing devices (e.g., the head-mounted display) have limited field-of-view (FoV), usually represented as the angular extent of the field that can be observed. To render the view, only part of the sphere is required. Such "sub-areas" would be useful in "tiling-like" schemes that can be used to support spatial-adaptive super-resolution over the 360-degree view. (Note that the "tile" here is not necessarily a rectangular-shaped tile.) That is, if only a small area on the sphere will be viewed by the user, we may only need to apply super-resolution to a sub-portion of the sphere instead of the full sphere. As a result, performing super-resolution on the full icosahedral mesh may no longer be necessary as it requires more computation resources for areas that are not watched by users.

To support both faster operation and super-resolution on a sub-portion of the sphere, we propose a partial refinement scheme to generate the "Focused Icosahedral Mesh". We first create a Level-1 icosahedral mesh by refining each face on a unit icosahedron into 4 faces. In this way, the 20-face icosahedron is refined into a Level-1 icosahedral mesh with 80 faces. An example full Level-1 mesh with 80 faces is shown in Figure 1(a). We then select one face out of the 80 faces of the Level-1 icosahedral mesh and only refine triangles located inside the selected Level-1 face.

Specifically, in our focused mesh representation, we select the face of the Level-1 mesh that covers the position of <latitude=0, longitude=0> on the sphere since very little distortion is introduced when pixels near this area are projected to the 2D plane. Figure 1(b) shows the Focused Level-2 mesh where the selected Level-1 face is refined into 4 smaller faces. Figures 1(c) and 1(d) show the Focused Level-3 and Focused Level-5 meshes, respectively.

3.1.1 Rotating content to the Focused Level-1 origin. While a full Level-1 mesh has 80 faces, we only generate one single Focused icosahedral mesh by refining one selected face, and our spherical super-resolution model only operates on this single Focused icosahedral mesh. To allow our model to perform super-resolution for any area on the sphere, we thus need to map spherical pixel content that belongs to any arbitrary full Level-1 mesh face to the face that is selected to be refined.



(a) **Full** Level-1 Mesh (b) **Focused** Level-2 Mesh (c) **Focused** Level-3 Mesh (d) **Focused** Level-5 Mesh

Fig. 1. Example of meshes in Level-1, Level-2, Level-3 and Level-5. To create "Focused icosahedral meshes", we select one face in the full Level-1 mesh and repeatedly refine triangles in this Level-1 face to obtain Focused Level-X meshes.

To do so, we pre-compute a rotation matrix $R \in \mathbb{R}^{N_F \times N_V \times C}$, where N_F represents the total number of faces in a full Level-1 mesh, which is 80, and N_V is the number of vertices in a Level-1 face, as shown in Figure 1. A Level-1 mesh consists of 80 triangles, each containing 3 vertices. C represents the number of dimensions of Euclidean coordinates in the sphere, namely xyz.

We denote the Level-1 face selected to be refined as face F_0. To rotate an arbitrary face $F_i, i \in (0, 80)$ on the Level-1 mesh to the refined face F_0, we need to find a rotation matrix R_i for face F_i such that $F_i = R_i \cdot F_0$, where F_i and F_0 are 3×3 matrices that represent the xyz coordinates of three vertices of a triangle face.

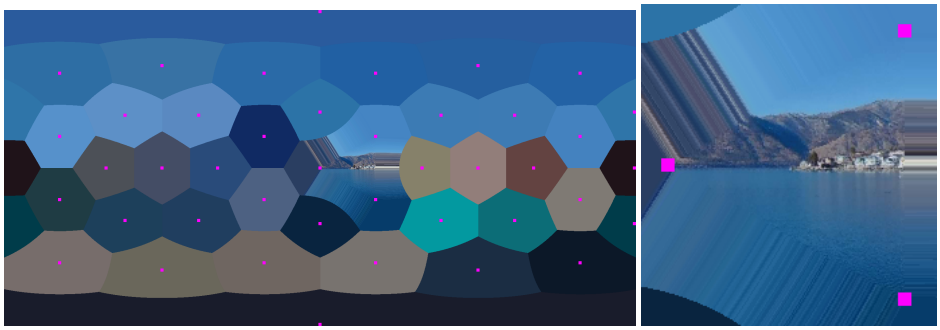
We can obtain R_i as: $R_i = F_i \cdot F_0^{(-1)}$. We first rotate the vertices in the Focused Level-X Mesh with the rotation matrix R, and then compute a mapping from each pixel in the input representation (e.g., an equirectangular image) to the rotated Focused Level-X vertex. Instead of refining all 80 faces in a mesh, in this way, we can represent all 80 different faces on the full Level-1 mesh through a single Focused Mesh file with just one face refined and other faces discarded. This allows us to save a significant amount of computation and storage resources and achieves better parameter efficiency.

Figure 2 visualizes how one focused icosahedral mesh can be used to represent all 80 different Level-1 faces. Figure 2(a) shows an original equirectangular-projected 360-degree image. In this image, we highlight two areas marked by magenta circles. In Figure 2(b), the left-hand-side image shows the Focused Level-9 mesh visualized on an equirectangular image. Magenta points in this figure represent vertices in the full Level-1 mesh. There are 42 vertices in the full Level-1 mesh. The right-hand-side image in Figure 2(b) magnifies the refined face in the Focused icosahedral mesh to show details. We can see that content in this face are in the same position as in the original equirectangular-projected image. Figure 2(c) shows the resulting visualization when we rotate a different Level-1 face to the refined face. The image on the right magnifies the refined face to show details.

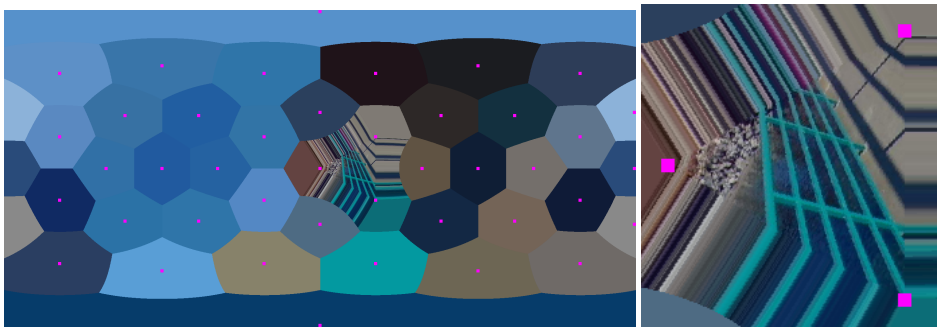
3.1.2 Mesh sizes. Table 1 shows the number of vertices in both Full and Focused icosahedral meshes in different levels of refinement. A Full Level-9 mesh has more than 2.6 million vertices and requires more than 1.9 GB of space for memory during inference and storage. On the other hand, a Focused Level-9 mesh has only about 33K vertices, requiring only about 31 MB memory storage space.

We know that the area of a unit spherical surface is 4π. A frame generated through the equirectangular projection covers a corresponding area of $2\pi \times \pi = 2\pi^2$. (The circumference of the equator is 2π.) Suppose there are N_x vertices in the Full Level-X mesh, given that vertices on the icosahedral mesh are roughly uniformly distributed on the sphere, we can estimate the equivalent 2D

(a) This figure shows an equirectangular-projected 360-degree image. Magenta circles, b and c, in this figure mark areas corresponding to two different refined faces.(Original photo by Timothy Oldfield on Unsplash: https://unsplash.com/photos/luufnHoChRU)



(b) The left-hand-side image displays the Focused Level-9 mesh visualized on an equirectangular image. The right-hand-side image displays a magnified view of the refined face (circle **b** in figure (a)). In both images, magenta points represent vertices in the full Level-1 mesh.



(c) This figure displays a different Level-1 icosahedral face (circle **c** in figure (a)) rotated to the face refined in the Focused Mesh. Pixel values from the original image are attached to rotated vertices by inverting the rotation for positions of the mesh vertices then finding the nearest neighbor pixel of this rotated position.

Fig. 2. Visualizing the Focused Icosahedral mesh.

Table 1. Number of vertices in **Full** icosahedral mesh, **Focused** icosahedral mesh, their corresponding memory and storage sizes, and their roughly-equivalent 2D planar resolution in the equirectangular projection.

Level	Level-6	Level-7	Level-8	Level-9
Full	40,962 (30 MB)	163,842 (118 MB)	655,362 (471 MB)	2,621,442 (1.9 GB)
Focused	600 (540 KB)	2,184 (2.0 MB)	8,424 (7.8 MB)	33,192 (31 MB)
2D planar	360x180	720x360	1440x720	2880x1440

equirectangular-projected frame resolution as follows: $W = \sqrt{N_x \times \pi}$, $H = W/2$, where W and H are the width and height of the equirectangular projection, respectively. The results are listed in Table 1. We find that Level-6 mesh is roughly equivalent to the 2D equirectangular projection in 360x180 resolution, and that Level-9 mesh is roughly equivalent to the 2D equirectangular projection in 2880x1440 resolution.

3.2 MeshConv

The MeshConv operation introduced by Jiang et al. [28] is performed by taking a linear combination of linear operator values computed on a set of input mesh vertex values. MeshConv can be formulated as follows:

$$\text{MeshConv}(F;\ \theta) = \theta_0 IF + \theta_1 \nabla_{lat} F + \theta_2 \nabla_{lng} F + \theta_3 \nabla^2 F, \tag{1}$$

where I represents the identity, which can be regarded as the $0th$ order differential, same as ∇_{00}. ∇_{lat} and ∇_{lng} are derivatives in two orthogonal spatial dimensions, which can be viewed as the $1st$ order differential. ∇^2 stands for the Laplacian operator, which can be considered as the $2nd$ order differential.

At a high level, these linear operators can be viewed as computing a set of local information near each vertex of the mesh. The standard 3×3 cross correlation operation can be viewed as a set of nine linear operators. Each of the linear operators returns a value of either the pixel itself or an adjacent pixel. Compared to the 3×3 convolution, it is clear that the set of four linear operators used by MeshConv is less expressive. They not only extract less information per pixel, but this information also can drop information about a vertex's surrounding. For example, the gradient operation on the mesh computes a 3-dimensional average of either six or seven values. Another degree-of-freedom is dropped from the gradient when taking only the east-west and north-south components of the gradient. We hypothesize that some of the information excluded from the linear operator computations could be useful for the super-resolution task. To attempt to mitigate this information loss, rather than including single MeshConv ops in the previous ResBlock architecture used in transposed MeshConv, we include pairs of composed MeshConv ops (as shown in Figure 3 ResBlock depiction) in our model. These paired operations aggregate more local information around a vertex before the non-linearity is applied, allowing the network to capture more useful characteristics needed for the super-resolution task.

3.3 VertexShuffle

MeshConv Transpose. To upscale the downsampled low level mesh in semantic segmentation task, UGSCNN [28] proposed a MeshConv Transpose operation. MeshConv Transpose takes Level-i mesh for input and outputs a Level-$(i+1)$ mesh. It can be described as follows:

$$M_{i+1} = \text{MeshConv}(\text{Padding}(M_i)), \tag{2}$$

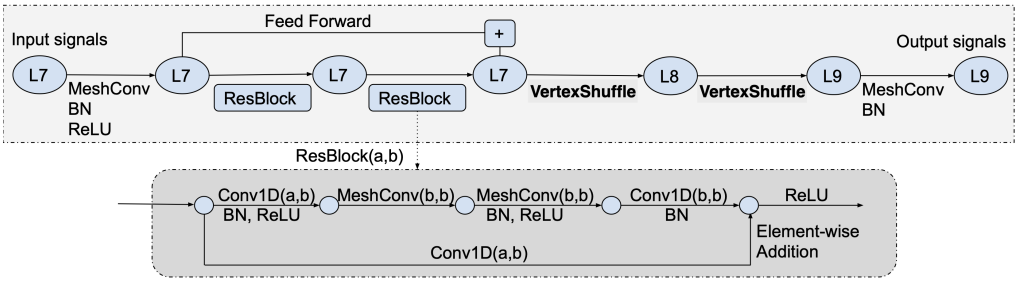


Fig. 3. Architecture of our proposed Spherical Super-Resolution (**SSR**) model that uses MeshConv and VertexShuffle. L7 represent the input Level-7 mesh, and L9 represents the output Level-9 mesh.

where Padding represents zero padding, M_{i+1} and M_i are Level-$(i+1)$ mesh and Level-i mesh, respectively. In general, MeshConv Transpose simply pads 0s on new vertices in Level-$(i+1)$ mesh, then applies a regular MeshConv on the new zero-padded Level-$(i+1)$ mesh. While this operation is easy to implement, as we will show in the Section 4, it is inefficient.

VertexShuffle. Motivated by PixelShuffle [47] commonly used in 2D super-resolution models, we propose a novel VertexShuffle operation to use in our spherical super-resolution model. The VertexShuffle operation can be described as follows:

$$M_{i+1} = \text{VertexShuffle}(M_i) \tag{3}$$
$$M_i = M_{i0}, M_{i1}, M_{i2}, M_{i3} \tag{3a}$$
$$N'_{ij} = \text{MidPoint}(M_{i(j+1)}), j \in \{0, 1, 2\} \tag{3b}$$
$$N'_i = concat(N'_{i0}, N'_{i1}, N'_{i2}) \tag{3c}$$
$$N_i = unique(N'_i) \tag{3d}$$
$$\text{VertexShuffle}(M_i) = concat(M_{i0}, N_i) \tag{3e}$$

The input of our basic VertexShuffle operation can be represented as $M_i \in \mathbb{R}^{F \times V_i}$, where F is the feature dimension in Level-i, and V_i represents the number of vertices of Level-i mesh. The output is $M_{i+1} \in \mathbb{R}^{F' \times V_{i+1}}$, where F' is the feature dimension in Level-$(i+1)$, which is $F/4$ in our work, and V_{i+1} represents the number of vertices of the Level-$(i+1)$ mesh.

We first split M_i into four parts $\{M_{i0}, M_{i1}, M_{i2}, M_{i3}\}$ along feature map dimension, where $M_{ij} \in \mathbb{R}^{F' \times V_i}$, $j = \{0, 1, 2, 3\}$ and $F' = F/4$. We keep M_{i0} as our Level-i mesh features, which will be used later. M_{i1}, M_{i2}, M_{i3} are used to refine features of vertices in Level-$(i+1)$ mesh.

As we introduced before, a spherical mesh can be obtained by progressively sub-dividing each face of the unit icosahedron into four equal triangles. Here, we treat a single triangle face as a sequence of vertices, v_0, v_1, v_2 and a sequence of edges $(v_0, v_1), (v_1, v_2), (v_2, v_0)$. The refinement process can be regarded as progressively constructing midpoint vertex on associated edges, and new edges in Level-$(i+1)$ are created between each pair of midpoint vertices, thus a single face in Level-i is refined into four new faces in Level-$(i+1)$.

To fully make use of feature maps in Level-i, we use M_{i1}, M_{i2}, M_{i3} to refine vertices in Level-$(i+1)$ mesh. Specifically, we use M_{i1} to calculate midpoint between (v_0, v_1), M_{i2} to calculate midpoint between (v_1, v_2), and M_{i3} to calculate midpoint between (v_2, v_0). Midpoint vertex values are constructed by averaging the values associated with the original two vertices on an edge. Thus,

Equation (3b) can be described as follows:

$$N'_{i0} = \text{MidPoint}(M_{i1}) = (M_{i1}(v_0) + M_{i1}(v_1))/2 \tag{4a}$$
$$N'_{i1} = \text{MidPoint}(M_{i2}) = (M_{i2}(v_1) + M_{i2}(v_2))/2 \tag{4b}$$
$$N'_{i2} = \text{MidPoint}(M_{i3}) = (M_{i3}(v_2) + M_{i3}(v_0))/2 \tag{4c}$$

Thus, we can get a set of midpoint vertices N'_i, which are new vertices generated in Level-$(i+1)$ mesh. However, there exist redundant midpoints due to the shared edges that may be calculated twice. We have to perform deduplication on the set of midpoint vertices. Here, we simply select the first instance of a midpoint. Then, we have a set of unique midpoint vertices that are used to refine the next level mesh $N_i \in \mathbb{R}^{F' \times A_i}$, where $A_i = V_{i+1} - V_i$. Finally, we concatenate partial feature map in Level-i, M_{i0}, with the new calculated midpoint vertices N_i to create our Level-$(i+1)$ mesh.

Compared to MeshConv Transpose, VertexShuffle does not have extra learnable parameters. Thus, our implementation of VertexShuffle is not only more parameters-efficient, but also achieves significantly better performance. The evaluation results are presented in Section 4.

3.4 VertexShuffle_V2

The super-resolution performance depends on how features of refined midpoints are generated. In this section, we describe a new approach we design for calculating features of vertices in the Level-$(i+1)$ mesh using feature maps in Level-i, i.e., $\{M_{i0}, M_{i1}, M_{i2}, M_{i3}\}$.

We refer to this new approach as VertexShuffle_V2. We note that compared to the original VertexShuffle operation, VertexShuffle_V2 can improve the performance without requiring additional parameters.

As before, the input of our basic VertexShuffle operation can be represented as $M_i \in \mathbb{R}^{F \times V_i}$, where F is the feature dimension in Level-i, and V_i is the number of vertices of Level-i mesh. The output is $M_{i+1} \in \mathbb{R}^{F' \times V_{i+1}}$. M_i is split into four parts $\{M_{i0}, M_{i1}, M_{i2}, M_{i3}\}$ along the feature map dimension. The new VertexShuffle_V2 operation can be described as follows:

$$M_{i+1} = \text{VertexShuffle\_V2}(M_i) \tag{5}$$
$$M_i = M_{i0}, M_{i1}, M_{i2}, M_{i3} \tag{5a}$$
$$M'_i = (M_{i0} + M_{i1})/2 \tag{5b}$$
$$L_i = (M_{i2} + M_{i3})/2 \tag{5c}$$
$$N'_{jk} = \text{MidPoint}_{jk}(L_i) \tag{5d}$$
$$N'_i = concat(N'_{01}, N'_{12}, N'_{20}) \tag{5e}$$
$$N_i = unique(N'_i) \tag{5f}$$
$$\text{VertexShuffle\_V2}(M_i) = concat(M'_i, N_i) \tag{5g}$$

Unlike Equation (3e) that directly uses M_{i0} as part of the mesh feature representation, we use the mean of these first two feature maps in level i, i.e., $\{M_{i0}, M_{i1}\}$ of our Level-i mesh, as shown in Equation (5g). We then compute the mean of the remaining two feature maps, $\{M_{i2}, M_{i3}\}$, to refine new vertices in Level-$(i+1)$ mesh. In this way, with VertexShuffle_V2, features of vertices in the refined Level-$(i+1)$ mesh are calculated using two feature maps in level i instead of just one as in VertexShuffle. Specifically, we calculate the mean value between M_{i2}, M_{i3} as L_i and use it to calculate the midpoints between (v_0, v_1), (v_1, v_2), (v_2, v_0). Midpoint vertex values are constructed by averaging the values associated with the original two vertices on a edge. Equation (5d) can be

described as follows:

$$N'_{01} = \text{MidPoint}_{01}(L_i) = (L_i(v_0) + L_i(v_1))/2 \tag{6a}$$

$$N'_{12} = \text{MidPoint}_{12}(L_i) = (L_i(v_1) + L_i(v_2))/2 \tag{6b}$$

$$N'_{20} = \text{MidPoint}_{20}(L_i) = (L_i(v_2) + L_i(v_0))/2 \tag{6c}$$

We perform deduplication via the same approach as Equation (3d) to obtain a set of unique midpoint vertices that can be used to refine the next level mesh, i.e., $N_i \in \mathbb{R}^{F' \times A_i}$. We then concatenate them with the averaged partial feature map in Level-i, M'_i, to create our new Level-$(i+1)$ mesh. We find that the performance of VertexShuffle_V2 improves over the original VertexShuffle method, while without introducing additional parameters.

3.5 Model architecture

We apply our Focused icosahedral mesh and VertexShuffle operation in the super-resolution task. The architecture of our spherical super-resolution (SSR) model is shown in Figure 3. In this figure, we show the input of our model as a Level-7 Focused icosahedral mesh, it first goes through a MeshConv layer with Batch Normalization [26] followed by a ReLU activation function. Then, we use two adapted Residual Blocks [25] to further extract features. The adapted residual blocks include two MeshConv layers (as discussed in Section 3.2). We concatenate the output of the first MeshConv and the output from the two ResBlocks by element-wise addition. After that, we use two VertexShuffle layers to upscale feature maps. Finally, our model ends up with a MeshConv layer, generating a Level-9 Focused icosahedral mesh with the correct number of channels.

3.6 Loss function

Similar to general super-resolution tasks, our goal is to minimize the loss between the reconstructed images Y_i and the corresponding ground truth high-resolution images H_i. Given a set of high-resolution images H_i and their corresponding low-resolution images X_i, we represent the loss as follows:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(H_i - Y_i)^2, \text{ where } Y_i = F(X_i) \tag{7}$$

$$\text{Loss} = 10 \times \log_{10}(\text{MSE}) \tag{8}$$

where N is the number of training samples. The loss function is the negative peak signal-to-noise ratio(PSNR), which aligns with our task to maximize the PSNR.

4 EVALUATION

4.1 Dataset

For evaluation, we used a publicly-available 360-degree video dataset [15]. This dataset contains 5 videos of 4K quality at the frame rate of 30 frames-per-second (fps). To use our Focused Icosahedral Mesh in training, for each frame in a video segment or a video (we describe a "per segment" and a "full video" training strategy in Section 4.2), we train for all 80 Level-1 faces. As discussed, the refined Level-1 face is selected as a face located on the equator to avoid distortion near pole areas. Contents on the remaining 79 faces are rotated to the refined Level-1 face during training and inference. We note that for our super-resolution tasks, the model is expected to overfit frames in the specific video segment or full video it is trained on. That is, the training data and the testing data are the same set of video frames.

4.2 Training Strategies

To evaluate the super-resolution performance in different scenarios, we consider 2 different upscaling factors in our experiments, ×4 and ×8. For upscaling factor ×4, the number of output vertices is ≈16x the number of input vertices. In this setting, the low-resolution input data is in Level-7, which is roughly equivalent to a 2D equirectangular-projected frame in 720x360 resolution. The high-resolution target data is in Level-9, which is roughly equivalent to 2880x1440 equirectangular-projected frames. For ×8, the number of output vertices is ≈64x the number of input vertices. With ×8, we use Level-6 mesh as input data, roughly equivalent to a 2D equirectangular frame in 360x180 resolution, and the high-resolution data is in Level-9.

We consider two real-world usage scenarios that correspond to two different training strategies. The first one is the "**per segment**" training strategy. In this case, we train a different super-resolution model for each video segment that is used in DASH-based video streaming. To setup the experiments for this scenario, for each video in the dataset, we use FFmpeg [5] to divide the full video temporally into segments of 1-second long each. For each segment, we train a super-resolution model for the 30 frames it contains. For each video in the dataset, we train the first video segment for 40 epochs, while for the remaining video segments, we take advantage of temporal locality of videos. That is, video frames that are close together temporally may be similar. We thus use a previous video segment's model as a pre-trained model to initialize parameters in the model for the current segment. This allows us to pursue training efficiency by training each video segment for only 15 epochs.

In addition, we consider the "**full video**" training strategy, where we can train a single model for an entire video. This strategy allows us to save the storage and network bandwidth for storing and transmitting models. For example, with per-segment training, we have to store and stream 60 models for a single 1-minute long video, while only one model is sufficient for the whole-video training strategy. A potential drawback of this approach, however, is that the quality of super-resolved frames may not be as high as the "per-segment" training strategy as the model is not fine-turned on each individual segment.

For both scenarios, we set the learning rate to 0.01 and use Adam [31] as our optimizer. We conducted our experiments on a desktop machine with Intel(R) Core(TM) i7-8700K CPU and GeForce RTX 2080 Ti GPU.

4.3 Baseline Models

We train models for all videos in the dataset using our proposed spherical super-resolution (SSR) model and compare its results with three baseline models: 2D super-resolution with Upsample, 2D super-resolution with PixelShuffle, and spherical super-resolution with transposed MeshConv (i.e., instead of VertexShuffle). The model architecture for transposed MeshConv is similar to our spherical super-resolution (SSR) model with the exception that it only has one MeshConv layer in its residual block. This is consistent with the original residual block used in Jiang [28]. The model architecture for the 2D baseline models are compatible with our spherical super-resolution (SSR) model: residual blocks in 2D models consist of two Conv2d operations. Our baseline models are all trained from scratch.

We focus our comparison on three aspects: visual quality (i.e., the PSNR value), model size, and inference time. For 2D baseline models, we tile the frames spatially into small patches. For example, in the Level-7 to Level-9 super-resolution experiments with upscaling factor ×4, we use the patch size of 45×45 pixels for input data, and thus, 180×180 pixels for output and target data. The upscaling factor is ×4, which is the same as our SSR model. Given that the Level-7 mesh is roughly equivalent to a 2D equirectangular-projected frame in 720x360 resolution, the total number of

Table 2. PSNR (dB) results for all 5 videos using the **per segment** training strategy with upscale factor ×4 in our testing dataset. Each segment is 1 second long.

Model	Diving	Timelapse	Venice	Paris	Rollercoaster	Average
2D with Upsample	30.34	31.75	35.90	28.00	25.31	30.26
2D with PixelShuffle	33.42	**34.13**	**40.07**	**32.45**	**33.62**	**34.74**
Spherical: MeshConv with transposed MeshConv	31.50	30.27	38.28	26.64	28.14	30.97
Spherical: MeshConv with VertexShuffle (**SSR**)	33.20	32.06	33.72	28.90	31.99	31.97
Spherical: MeshConv with VertexShuffle_V2 (**SSR2**)	**33.53**	31.76	34.38	28.94	32.96	32.31

Table 3. PSNR (dB) results for all 5 videos using the **full video** training strategy with upscaling factor ×4 in our testing dataset. The length of each video (in seconds) is shown in the second row of the table.

Model	Diving 61s	Timelapse 74s	Venice 80s	Paris 55s	Rollercoaster 61s	Average
2D with Upsample	30.08	30.31	33.76	30.95	23.13	29.65
2D with PixelShuffle	30.38	30.71	**37.92**	**31.29**	29.84	32.03
Spherical: MeshConv with transposed MeshConv	31.63	29.75	35.81	26.82	29.11	30.62
Spherical: MeshConv with VertexShuffle_V2 (**SSR2**)	**34.38**	**31.69**	33.87	29.22	**33.50**	**32.53**

Table 4. PSNR (dB) results for all 5 videos using the **full video** training strategy with upscaling factor ×8 in our testing dataset.

Model	Diving	Timelapse	Venice	Paris	Rollercoaster	Average
2D with Upsample	25.72	25.02	30.94	24.15	16.25	24.42
2D with PixelShuffle	29.77	**29.04**	**40.10**	**30.84**	27.25	**31.40**
Spherical: MeshConv with VertexShuffle_V2 (**SSR2**)	**31.93**	28.67	31.32	28.44	**30.20**	30.11

patches is $16 \times 8 = 128$. For the Level-6 to Level-9 experiments, since the 2D equivalent input frame resolution is 360x180, the total number of patches is $8 \times 4 = 32$ using the same 45×45 patch size.

4.4 PSNR Results

Per segment training with upscaling factor ×4. The mean PSNR values obtained by all models for each video using the **per segment** training strategy with upscaling factor ×4 are shown in Table 2. 2D baseline model with PixelShuffle achieves the best PSNR value because of the efficiency of 2D convolution. Compared to the spherical baseline model that uses transposed MeshConv operation, our SSR model achieves better results for 4 out of 5 videos as well as on average. This shows the improvement of our new VertexShuffle operation. In addition, the results also show that our new VertexShuffle_V2 operation (shown as **SSR2**) outperforms **SSR** for all but one videos. This shows that calculating features of refined midpoint vertices by using more than one feature maps can achieve improved performance compared to using only one feature map for each vertex.

Full video training with upscaling factor ×4. For the **full video** training strategy, as only one model is learned for the entire video (e.g., over 60 seconds long), the super-resolution performance degrades for many videos and models. The PSNR results for each video are shown in Table 3. The second row in the table shows the length of each video in the dataset. Results show that our **SSR2** model performs the best among all models for 3 out of 5 videos and achieves the best average performance. It is also worth noting that **SSR2**'s performance with **full video** training is

Table 5. Comparison of total number of model parameters, model storage size, and per-frame inference time.

Model	Total # of Parameters	Storage Size	Per-frame Inference Time
2D with Upsample	86,163	343 KB	23.39 ms
2D with PixelShuffle	84,948	338 KB	13.54 ms
Spherical: MeshConv with transposed MeshConv	64,265	273 KB	997.91 ms
Spherical: MeshConv with VertexShuffle (**SSR**)	79,925	333 KB	74.35 ms

comparable to **per segment** training. Given that only one model is trained for the full duration of the video, this indicates **SSR2** can substantially save the storage size and network bandwidth for transmitting the super-resolution models.

Full video training with upscaling factor ×8. With a higher upscaling factor of ×8, it is more challenging to perform super-resolution. The mean PSNR results obtained for each video with full video training strategy are shown in Table 4. We were unable to complete the experiments for the "Spherical: MeshConv with transposed MeshConv" baseline model due to its inefficiency. We thus do not show the results for this baseline model.

Results show that the performance of our **SSR2** model is still comparable to the 2D baseline model with PixelShuffle. We note that the performance gain of "2D with PixelShuffle" is mainly due to the "Venice" video. For the remaining four videos, on average, our **SSR2** model outperforms the 2D PixelShuffle model with an average PSNR of 29.81 dB (SSR2) vs. 29.23 dB (2D PixelShuffle).

4.5 Model Parameters, Storage Sizes, and Inference Time

Table 5 shows the total number of parameters of all models and their corresponding storage sizes. The spherical baseline model with transposed MeshConv operations used as one of our baseline models has a smaller model size due to the fewer MeshConv layers in its residual block. However, its inference time is significantly longer than all other models. The numbers for the **SSR2** model is the same as **SSR**, and we omit them from the table.

In our spherical models, we divide the original omnidirectional frame into 80 faces. In 2D baseline models, the number of tiled patches, however, is different from the number of faces (128 patches for the ×4 scenario and 32 for the ×8 scenario). Thus, it is not fair to simply compare the inference time for each face or tile/patch. Instead, we use the per-frame inference time for comparison. The experiment results are shown in Table 5. 2D baseline models have the fastest inference time. Our SSR model is slower than 2D baseline models but significantly faster than the spherical baseline with transposed MeshConv operations. The speed of 2D models can be attributed to the efficient cuDNN library that the PyTorch 2D convolution operation uses. On the other hand, our SSR model is implemented based on the PyTorch library only. In addition, given that a user will only watch a sub-portion of the spherical frame in real scenarios, we do not need to perform super-resolution for all 80 faces of a frame. This indicates our SSR model can be performed in real-time to meet the 30 fps frame rate required by most videos.

5 CONCLUSION

In this paper, we proposed a spherical super-resolution model – SSR. SSR directly operates on spherical signals, which can avoid issues in applying 2D super-resolution to spherical data, such as distortion, oversampled pixels, etc. We created a memory- and bandwidth-efficient representation of the spherical mesh – the Focused Icosahedral Mesh, which is more flexible than full meshes and saves a significant amount of computation resources. We created a novel VertexShuffle operation, inspired by the PixelShuffle operation for 2D super-resolution, and an improved VertexShuffle_V2

operation to further improve our model. We conducted comprehensive experiments on our SSR with two different training strategies and two upscaling factors, ×4 and ×8. Results show that our model outperforms other baseline models in most videos.

6 FUTURE WORK

In this work, we treated frames in the 360-degree videos as standalone frames without considering their inherent relationships. In the future, we plan to consider adding temporal information across frames, based on works such as [6, 9, 38, 48]. For example, we can take motion vector into consideration to capture the temporal information [34, 45]. We also plan to deploy SSR and VertexShuffle operations in real-world scenarios for real-time 360-degree video super-resolution.

ACKNOWLEDGMENTS

We appreciate constructive comments from anonymous referees. This work is partially supported by NSF under grants CNS-2200042 and CNS-2200048.

REFERENCES

[1] 2016. Next-generation video encoding techniques for 360 video and VR. https://code.facebook.com/posts/1126354007399553/next-generation-video-encoding-techniques-for-360-video-and-vr/.
[2] 2017. EAC. https://blog.google/products/google-ar-vr/bringing-pixels-front-and-center-vr-video/.
[3] 2017. End-to-end optimizations for dynamic streaming. https://code.facebook.com/posts/637561796428084/end-to-end-optimizations-for-dynamic-streaming/.
[4] 2022. Equirectangular Projection. http://mathworld.wolfram.com/EquirectangularProjection.html.
[5] 2022. FFmpeg. http://www.ffmpeg.org/.
[6] Arbind Agrahari Baniya, Tsz-Kwan Lee, Peter W Eklund, and Sunil Aryal. 2023. Omnidirectional Video Super-Resolution using Deep Learning. *IEEE Transactions on Multimedia* (2023).
[7] Yanan Bao, Huasen Wu, Tianxiao Zhang, Albara Ah Ramli, and Xin Liu. 2016. Shooting a moving target: Motion-prediction-based transmission for 360-degree videos. In *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 1161–1170.
[8] John R Baumgardner and Paul O Frederickson. 1985. Icosahedral discretization of the two-sphere. *SIAM J. Numer. Anal.* 22, 6 (1985), 1107–1115.
[9] Jose Caballero, Christian Ledig, Andrew Aitken, Alejandro Acosta, Johannes Totz, Zehan Wang, and Wenzhe Shi. 2017. Real-time video super-resolution with spatio-temporal networks and motion compensation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4778–4787.
[10] Mingdeng Cao, Chong Mou, Fanghua Yu, Xintao Wang, Yinqiang Zheng, Jian Zhang, Chao Dong, Gen Li, Ying Shan, Radu Timofte, et al. 2023. NTIRE 2023 Challenge on 360deg Omnidirectional Image and Video Super-Resolution: Datasets, Methods and Results. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1731–1745.
[11] Jiawen Chen, Miao Hu, Zhenxiao Luo, Zelong Wang, and Di Wu. 2020. SR360: boosting 360-degree video streaming with super-resolution. In *Proceedings of the 30th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. 1–6.
[12] Zheng Chen, Yulun Zhang, Jinjin Gu, Linghe Kong, Xiaokang Yang, and Fisher Yu. 2023. Dual aggregation transformer for image super-resolution. In *Proceedings of the IEEE/CVF international conference on computer vision*. 12312–12321.
[13] Taco Cohen, Mario Geiger, Jonas Köhler, and Max Welling. 2017. Convolutional networks for spherical signals. *arXiv preprint arXiv:1709.04893* (2017).
[14] Taco S Cohen, Maurice Weiler, Berkay Kicanaoglu, and Max Welling. 2019. Gauge equivariant convolutional networks and the icosahedral cnn. *arXiv preprint arXiv:1902.04615* (2019).
[15] Xavier Corbillon, Francesca De Simone, and Gwendal Simon. 2017. 360-degree video head movement dataset. In *Proceedings of the 8th ACM on Multimedia Systems Conference*. 199–204.
[16] Xavier Corbillon, Gwendal Simon, Alisa Devlic, and Jacob Chakareski. 2017. Viewport-adaptive navigable 360-degree video delivery. In *Communications (ICC), 2017 IEEE International Conference on*. IEEE, 1–7.
[17] Mallesham Dasari, Arani Bhattacharya, Santiago Vargas, Pranjal Sahu, Aruna Balasubramanian, and Samir R Das. 2020. Streaming 360-degree videos using super-resolution. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 1977–1986.

[18] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. 2014. Learning a deep convolutional network for image super-resolution. In *European conference on computer vision*. Springer, 184–199.

[19] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. 2015. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence* 38, 2 (2015), 295–307.

[20] Chao Dong, Chen Change Loy, and Xiaoou Tang. 2016. Accelerating the super-resolution convolutional neural network. In *European conference on computer vision*. Springer, 391–407.

[21] Marc Eder, Mykhailo Shvets, John Lim, and Jan-Michael Frahm. 2020. Tangent Images for Mitigating Spherical Distortion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12426–12434.

[22] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[23] Mario Graf, Christian Timmerer, and Christopher Mueller. 2017. Towards Bandwidth Efficient Adaptive Streaming of Omnidirectional Video over HTTP: Design, Implementation, and Evaluation. In *Proceedings of the 8th ACM on Multimedia Systems Conference*. ACM, 261–271.

[24] Yu Guan, Chengyuan Zheng, Xinggong Zhang, Zongming Guo, and Junchen Jiang. 2019. Pano: Optimizing 360 video streaming with a better understanding of quality perception. In *Proceedings of the ACM Special Interest Group on Data Communication*. 394–407.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[26] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[27] Michal Irani and Shmuel Peleg. 1991. Improving resolution by image registration. *CVGIP: Graphical models and image processing* 53, 3 (1991), 231–239.

[28] Chiyu Jiang, Jingwei Huang, Karthik Kashinath, Philip Marcus, Matthias Niessner, et al. 2019. Spherical cnns on unstructured grids. *arXiv preprint arXiv:1901.02039* (2019).

[29] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. 2016. Accurate image super-resolution using very deep convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1646–1654.

[30] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. 2016. Deeply-recursive convolutional network for image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1637–1645.

[31] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[32] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. 2017. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4681–4690.

[33] Na Li and Yao Liu. 2022. Applying VertexShuffle toward 360-degree video super-resolution. In *Proceedings of the 32nd Workshop on Network and Operating Systems Support for Digital Audio and Video*. 71–77.

[34] Sheng Li, Fengxiang He, Bo Du, Lefei Zhang, Yonghao Xu, and Dacheng Tao. 2019. Fast spatio-temporal residual network for video super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 10522–10531.

[35] Zheyuan Li, Yingqi Liu, Xiangyu Chen, Haoming Cai, Jinjin Gu, Yu Qiao, and Chao Dong. 2022. Blueprint separable residual network for efficient image super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 833–843.

[36] Jingyun Liang, Jiezhang Cao, Guolei Sun, Kai Zhang, Luc Van Gool, and Radu Timofte. 2021. Swinir: Image restoration using swin transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*. 1833–1844.

[37] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. 2017. Enhanced deep residual networks for single image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 136–144.

[38] Ding Liu, Zhaowen Wang, Yuchen Fan, Xianming Liu, Zhangyang Wang, Shiyu Chang, and Thomas Huang. 2017. Robust video super-resolution with learned temporal dynamics. In *Proceedings of the IEEE International Conference on Computer Vision*. 2507–2515.

[39] Zhenxiao Luo, Baili Chai, Zelong Wang, Miao Hu, and Di Wu. 2023. Masked360: Enabling Robust 360-degree Video Streaming with Ultra Low Bandwidth Consumption. *IEEE Transactions on Visualization and Computer Graphics* 29, 5 (2023), 2690–2699.

[40] Anahita Mahzari, Afshin Taghavi Nasrabadi, Aliehsan Samiei, and Ravi Prakash. 2018. FoV-Aware Edge Caching for Adaptive 360Â° Video Streaming. In *2018 ACM Multimedia Conference on Multimedia Conference*. ACM, 173–181.

[41] Afshin Taghavi Nasrabadi, Anahita Mahzari, Joseph D Beshay, and Ravi Prakash. 2017. Adaptive 360-degree video streaming using scalable video coding. In *Proceedings of the 2017 ACM on Multimedia Conference*. ACM, 1689–1697.

[42] Akito Nishiyama, Satoshi Ikehata, and Kiyoharu Aizawa. 2021. 360 single image super resolution via distortion-aware network and distorted perspective images. In *2021 IEEE International Conference on Image Processing (ICIP)*. IEEE, 1829–1833.

[43] Stefano Petrangeli, Viswanathan Swaminathan, Mohammad Hosseini, and Filip De Turck. 2017. An HTTP/2-Based Adaptive Streaming Framework for 360 Virtual Reality Videos. In *Proceedings of the 2017 ACM on Multimedia Conference*. ACM, 306–314.

[44] Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. 2018. Flare: Practical Viewport-Adaptive 360-Degree Video Streaming for Mobile Devices. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. ACM, 99–114.

[45] Mehdi SM Sajjadi, Raviteja Vemulapalli, and Matthew Brown. 2018. Frame-recurrent video super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 6626–6634.

[46] Ayush Sarkar, John Murray, Mallesham Dasari, Michael Zink, and Klara Nahrstedt. 2021. L3BOU: Low Latency, Low Bandwidth, Optimized Super-Resolution Backhaul for 360-Degree Video Streaming. In *2021 IEEE International Symposium on Multimedia (ISM)*. IEEE, 138–147.

[47] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. 2016. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1874–1883.

[48] Yuki Shiba, Satoshi Ono, Ryo Furukawa, Shinsaku Hiura, and Hiroshi Kawasaki. 2017. Temporal shape super-resolution by intra-frame motion encoding using high-fps structured light. In *Proceedings of the IEEE International Conference on Computer Vision*. 115–123.

[49] Liyang Sun, Fanyi Duanmu, Yong Liu, Yao Wang, Yinghua Ye, Hang Shi, and David Dai. 2018. Multi-path multi-tier 360-degree video streaming in 5G networks. In *Proceedings of the 9th ACM Multimedia Systems Conference*. ACM, 162–173.

[50] Ying Tai, Jian Yang, and Xiaoming Liu. 2017. Image super-resolution via deep recursive residual network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3147–3155.

[51] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[52] Xiaoyan Wang, Tho Duc Nguyen, Chanh Minh Tran, Eiji Kamioka, and Tan Xuan Phan. 2023. Central Vision based Super-resolution for 360-Degree Videos. In *Proceedings of the 2023 7th International Conference on Big Data and Internet of Things*. 34–39.

[53] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. 2018. Cbam: Convolutional block attention module. In *Proceedings of the European conference on computer vision (ECCV)*. 3–19.

[54] Lan Xie, Zhimin Xu, Yixuan Ban, Xinggong Zhang, and Zongming Guo. 2017. 360ProbDASH: Improving QoE of 360 Video Streaming Using Tile-based HTTP Adaptive Streaming. In *Proceedings of the 2017 ACM on Multimedia Conference*. ACM, 315–323.

[55] Alireza Zare, Alireza Aminlou, Miska M Hannuksela, and Moncef Gabbouj. 2016. HEVC-compliant tile-based streaming of panoramic video for virtual reality applications. In *Proceedings of the 2016 ACM on Multimedia Conference*. ACM, 601–605.

[56] Chao Zhang, Stephan Liwicki, William Smith, and Roberto Cipolla. 2019. Orientation-aware semantic segmentation on icosahedron spheres. In *Proceedings of the IEEE International Conference on Computer Vision*. 3533–3541.

[57] Xindong Zhang, Hui Zeng, Shi Guo, and Lei Zhang. 2022. Efficient long-range attention network for image super-resolution. In *European Conference on Computer Vision*. Springer, 649–667.

[58] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. 2018. Residual dense network for image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2472–2481.

[59] Chao Zhou, Zhenhua Li, and Yao Liu. 2017. A measurement study of oculus 360 degree video streaming. In *Proceedings of the 8th ACM on Multimedia Systems Conference*. ACM, 27–37.

[60] Chao Zhou, Zhenhua Li, Joe Osgood, and Yao Liu. 2018. On the effectiveness of offset projections for 360-degree video streaming. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 14, 3s (2018), 1–24.

[61] Chao Zhou, Shuoqian Wang, Mengbai Xiao, Sheng Wei, and Yao Liu. 2020. Adap-360: User-adaptive area-of-focus projections for bandwidth-efficient 360-degree video streaming. In *Proceedings of the 28th ACM International Conference on Multimedia*. 3715–3723.

[62] Yupeng Zhou, Zhen Li, Chun-Le Guo, Song Bai, Ming-Ming Cheng, and Qibin Hou. 2023. SRFormer: Permuted Self-Attention for Single Image Super-Resolution. *arXiv preprint arXiv:2303.09735* (2023).