

[DESCRIPTION]**[Title of Invention]****METHOD FOR COMPRESSING POINT CLOUD****[Technical Field]****[0001]**

The invention relates generally to compressing and representing point clouds, and more particularly to methods and system for predicting and applying transforms to three dimensional blocks of point cloud data for which some positions in a block may not be occupied by a point.

[Background Art]**[0002]****Point Clouds**

A point cloud is a set of data points in some coordinate system. In a three-dimensional coordinate (3D) system, the points can represent an external surface of an object. Point clouds can be acquired by a 3D sensor. The sensors measure a large number of points on the surface of the object, and output the point cloud as a data file. The point cloud represents the set of points that the device has measured.

[0003]

Point clouds are used for many purposes, including 3D models for manufactured parts, and a multitude of visualization, animation, rendering applications.

[0004]

Typically, the point cloud is a set of points in three-dimensional (3D) space, with attributes associated with each point. For example, a given point can have a specific (x, y, z) coordinate specifying its position, along with one or more attributes associated with that point. Attributes can include data such as color values, motion vectors, surface normal vectors, and connectivity information. The

amount of data associated with the point cloud can be massive, in the order of many gigabytes. Therefore, compression is needed to efficiently store or transmit the data associated with the point cloud for practical applications.

[0005]

Compression

A number of methods are known for compressing images and videos using prediction and transforms. Existing methods for compressing images and videos typically operate on blocks of pixels. Given a block of data for images or video, every position in the block corresponds to a pixel position in the image or video.

[0006]

However, unlike images or videos, if a 3D point cloud is partitioned into blocks, not all positions in the block are necessarily occupied by a point. Methods such as prediction and transforms used to efficiently compress video and image blocks will not work directly on blocks of 3D point cloud data. Therefore, there is a need for methods to perform prediction and transforms on blocks of 3D point cloud data for which some of the positions in the blocks may not be occupied by point data.

[0007]

Applications

With the recent advancements and reductions in cost of 3D sensor technologies, there has been an increasingly wide proliferation of 3D applications such as virtual reality, mobile mapping, scanning of historical artifacts, and 3D printing. These applications use different kinds of sensors to acquired data from the real world in three dimensions, producing massive amounts of data. Representing these kinds of data as 3D point clouds has become a practical method for storing and conveying the data independent of how the data are acquired.

[0008]

Usually, the point cloud is represented a set of coordinates or meshes indicating the position of each point, along with the one or more attributes associated with each point, such as color. Point clouds that include connectivity information among vertices are known as *structured* or *organized* point clouds. Point clouds that contain positions without connectivity information are *unstructured* or *unorganized* point clouds.

[0009]

Much of the earlier work in reducing the size of point clouds, primarily structured, has come from computer graphics applications. Many of those applications achieve compression by reducing the number of vertices in triangular or polygonal meshes, for example by fitting surfaces or splines to the meshes. Block-based and hierarchical octree-based approaches can also be used to compress point clouds. For example, octree representations can be used to code structured point clouds or meshes

[0010]

Significant progress has been made over the past several decades on compressing images and videos. The Joint Photographic Experts Group (JPEG) standard, H.264 or the Moving Picture Experts Group (MPEG-4) Part 10, also known as the Advanced Video Coding (MPEG-4 AVC) standard, and the High Efficiency Video Coding (HEVC) standard are widely used to compress images and video. These coding standards also utilize block-based and/or hierarchical methods for coding pixels. Concepts from these image and video coders have also been used to compress point clouds.

[Summary of Invention]

[0011]

The embodiments of the invention provide method and system for compressing a three-dimensional (3D) point cloud using prediction and

transformation of attributes of the 3D point cloud. The point cloud is partitioned into 3D blocks. To compress each block, projections of attributes in previously-coded blocks are used to determine directional predictions of attributes in the block currently being coded.

[0012]

A modified shape-adaptive transform is used to transform the attributes in the current block or the prediction residual block. The residual block results from determining a difference between the prediction block and the current block. The shape-adaptive transform is capable of operating on blocks that have “missing” elements or “holes.” i.e., not all possible positions in the block are occupied by points.

[0013]

As defined herein, the term “position” to refer to the location of a point in 3D space, i.e., the (x, y, z) location of a point in space, anywhere in space, not necessarily aligned to a grid. For example, the position can be specified by a floating-point number. The term “element” to refer to data at a position within a uniformly-partitioned block of data, similar in concept to how a matrix contains a grid of elements, or a block of pixels contains a grid of pixels.

[0014]

Two embodiments for handling holes inside shapes are provided. One embodiment inserts a value into each hole, and another example shifts subsequent data to fill the holes. A decoder, knowing the coordinates of the points, can reverse these processes without the need for signaling additional shape or region information in the compressed bitstream, unlike the prior-art shape adaptive discrete cosine transform (SA-DCT).

[Brief Description of the Drawings]

[0015]

[Fig. 1]

Fig. 1 is a block diagram of preprocessing a point cloud according to embodiments of the invention.

[Fig. 2A]

Fig. 2A is a block diagram of predicting points in a current block from points contained in non-empty adjacent blocks according to embodiments of the invention.

[Fig. 2B]

Fig. 2B is a schematic of a 3D point cloud block prediction method according to embodiments of the invention.

[Fig. 3A]

Fig. 3A is a schematic of a shape-adaptive discrete cosine transform process according to embodiments of the invention.

[Fig. 3B]

Fig. 3B is a schematic of an alternative shape-adaptive discrete cosine transform process according to embodiments of the invention.

[Fig. 4A]

Fig. 4A is a schematic of a graph transform formed by connecting adjacent points present in the 3D block according to embodiments of the invention.

[Fig. 4B]

Fig. 4B is an adjacency matrix **A** including weights associated with the adjacent points according to embodiments of the invention.

[Fig. 5]

Fig. 5 is a block diagram of the preprocessing and coding method according to embodiments of the invention.

[Fig. 6]

Fig. 6 is a block diagram of a decoding method according to embodiments of

the invention.

[Description of Embodiments]

[0016]

The embodiments of the invention provide a method and system for compressing a three-dimensional (3D) point cloud using prediction and transformation of attributes of the 3D point cloud.

[0017]

Point cloud preprocessing and block partitioning

Sometimes, point clouds are already arranged in a format that is amenable to block processing. For example, graph transforms can be used for compressing point clouds that are generated by sparse voxelization. The data in these point clouds are already arranged on a 3D grid where each direction has dimensions 2^j with j being a level within a voxel hierarchy, and the points in each hierarchy level have integer coordinates.

[0018]

Partitioning such a point cloud into blocks, where the points are already arranged on a hierarchical integer grid, is straightforward. In general, however, point clouds acquired using other techniques can have floating-point coordinate positions, not necessarily arranged on a grid.

[0019]

In order to be able to process point clouds without constraints on the acquisition technique, we preprocess the point cloud data so the points are located on a uniform grid. This preprocessing can also serve as a form of down-sampling.

[0020]

Fig. 1 is a block diagram of preprocessing 100 a point cloud 101. The point cloud can be acquired without any constraints of the acquisition modality. In one embodiment, the point cloud 101 is acquired by a depth sensor or scanner 103.

Alternatively, the point cloud can be acquired by multiple still cameras, or a video camera at different viewpoints. It is particularly noted that the amount of data can be extremely large, e.g., about several gigabytes or more, making storing and transmitting the data for practical applications difficult with conventional techniques. Hence, the data are compressed as described herein.

[0021]

The first step of preprocessing converts 110 the point cloud to an octree representation of voxels, also known as a 3D block of pixels, according to an octree resolution r 102, i.e., a size of edges of the voxels. Given the minimal octree resolution r , the point cloud is organized or converted 110 into octree nodes. If a node contains no points, then the node is removed from the octree. If a node contains one or more points, then the node is further partitioned into smaller nodes. This process continues until the size, or edge length of a leaf node reaches the minimal octree resolution r .

[0022]

Each leaf node corresponds to a point output by the partitioning step. The position of the output point is set to a geometric center of the leaf node, and the value of any attribute associated with the point is set 120 to an average value of one or more points in the leaf node. This process ensures that the points output by the preprocessing are located on a uniform 3D grid 140 having the resolution r .

[0023]

When the points are arranged on a uniform grid, the region encompassing the set of points is partitioned 160 into 3D blocks of size $k \times k \times k$. A block contains k^3 elements, however, many of these elements can be empty, unless the point cloud happens to contain points at every possible position in each block. A block may also have different numbers of elements in each direction; for example, a block can have dimensions $k \times m \times n$, hence containing $k \times m \times n$ elements.

[0024]

At this stage, the difference between these 3D point cloud blocks and 2D blocks of pixels from conventional image processing becomes apparent. In conventional image processing, all elements of each 2D block correspond to pixel positions present in the image. In other words, all blocks are fully occupied.

[0025]

However, in the block-based point cloud processing as described herein, the 3D blocks are not necessarily fully occupied. The blocks can contain between 1 and k^3 elements. Therefore, procedures, such as intra prediction and block-based transforms, used for conventional image and video coding cannot be directly applied to these 3D blocks. Hence, we provide techniques for accommodating the empty elements.

[0026]

We define 130 replacement point positions at the center of each octree leaf node. Thus, the preprocessed point cloud 140 has a set of attribute values and a set of point positions. The point cloud can now be partitioned 160 into the array $k \times k \times k$ blocks 170 according to a block edge size 150.

[0027]

Intra prediction of 3D point cloud blocks

Using prediction among blocks to reduce redundancy is a common technique in current coding standards such as H.264/AVC and HEVC. Adjacent decoded blocks are used to predict pixels in the current block, and then the prediction error or residuals are optionally transformed and coded in a bitstream. We describe a block prediction scheme using a low-complexity prediction architecture in which the prediction is obtained from three directions, i.e., (x, y, z) .

[0028]

As shown in Fig. 2A, points in a current block 201 can be predicted from

points contained in non-empty adjacent blocks 202, 203, and 204, when adjacent blocks are available. The point cloud encoder performs prediction in the x, y, and z directions and selects the prediction direction that yields the least distortion. Coding the current block without prediction from adjacent blocks can also be considered if that can yield a lower distortion. Therefore, the current block has the option of being coded with or without prediction.

[0029]

As described above, many of the k^3 elements in a block may not be occupied by points. Moreover, points within a block may not necessarily be positioned along the edges or boundaries of the block. The intra prediction techniques of H.264/AVC and HEVC use pixels along the boundaries of adjacent blocks to determine predictions for the current block.

[0030]

As shown in Fig. 2B for our 3D point cloud block prediction method, we use multivariate interpolation and extrapolation to determine a projection of the attribute values in, e.g., the adjacent block 202 onto the adjacent edge plane of the current block 201. For example, we project 205 of points onto top of current block, and project 206 points to the interior of the current block.

[0031]

Here, data from known points is used to determine an interpolation or prediction located at an arbitrary point, in this case, along the boundary between the previous block and the current block.

[0032]

In our case, suppose the block 202 above the current block contains a set of point positions $\mathbf{P} = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_N\}$, with the points having associated attribute values $\mathbf{A} = \{\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_N\}$. Given a point position along the boundary $\mathbf{p}_{\text{boundary}}$, the prediction takes the form

$$\mathbf{a}_{\text{boundary}} = f(\mathbf{P}, \mathbf{A}, \mathbf{p}_{\text{boundary}}),$$

where $\mathbf{a}_{\text{boundary}}$ is the predicted value of the attribute at the boundary.

[0033]

We can use a nearest-neighbor interpolation and extrapolation, which reduces complexity and simplifies the handling of degenerate cases in which the adjacent block contains only one or two points, or when all the points in the adjacent block are aligned on a plane perpendicular to the projection plane.

[0034]

After the attribute values along the boundary plane are estimated, these values are then projected 206 or replicated into the current block parallel to the direction of prediction. This is similar to how prediction values are replicated into the current block for the directional intra prediction used in standards such as H.264/AVC and HEVC.

[0035]

The projected and replicated values are used to predict attributes for points in the current block. For example, if the adjacent block in the y direction is used for prediction, then the set of points along the boundary $\mathbf{p}_{\text{boundary}}$ are indexed in two dimensions, i.e. $p(x, z)$, and the attribute for a point the current block $p_{\text{curr}}(x, y, z)$ is predicted using $a_{\text{boundary}}(x, z)$ for all values of y .

[0036]

Transforms for 3D block data

After the prediction process, a 3D block containing prediction residuals for each point in the current block, or the current block itself if it yields lower coding distortion, is transformed. As was the case for the prediction process, not all the positions in the block may be occupied by a point. Therefore, the transform is designed so that it will work on these potentially sparse blocks. We consider two

types of transforms: a novel variant of a conventional shape-adaptive discrete cosine transform (SA-DCT) designed for 3D point cloud attribute compression, and a 3D graph transform.

[0037]

Modified shape-adaptive DCT

The shape-adaptive DCT (SA-DCT) is a well-known transform designed to code arbitrarily shaped regions in images. A region is defined by a contour, e.g., around a foreground region of an image. All the pixels inside the region are shifted and then transformed in two dimensions using orthogonal DCTs of varying lengths. The contour positions and quantized transform coefficients are then signaled in the bitstream.

[0038]

For our 3D point cloud compression method, we treat the presence of points in a 3D block as a “region” to be coded, and positions in the block that do not contain points are considered as being outside the region. For the attribute coding application described herein, the point positions are already available at the decoder irrespective of what kind of transform is used.

[0039]

Because our 3D SA-DCT regions are defined by the point positions and not by the attribute values of the points, there is no need to perform operations, such as foreground and background segmentation and coding of contours, as is typically done when the SA-DCT is used for conventional 2D image coding.

[0040]

Fig. 3A shows our modified SA-DCT process, where closed circles 311 represent points in the point cloud, X 312 represent empty positions, and open circles 313 represent “filler” value for input to the DCT. Given a 3D block 301 of attribute values or prediction residual values, the points present in the block are

shifted 302 line by line along dimension 1 toward the border so that there are no empty positions in the block along that border, except for empty lines. We apply 303 a 1D DCT along the same direction. Then, we repeat 304-305 the shift and transform process on coefficients along dimensions 2 and 3 resulting in one DC and one or more AC coefficients. If there are empty positions between the first and last points in the column, we insert filler values, e.g. zero. Compression is achieved by quantizing the coefficients.

[0041]

Fig. 3B shows an alternative method that shifts 320 the remaining data in the column into those empty positions to eliminate interior empty positions, thus reducing the lengths of the DCTs.

[0042]

In another embodiment, all remaining empty positions in a 3D block are filled with predetermined values, so that all 1D DCTs applied to the block in a given direction have the same length, equal to the number of missing and non-missing elements along that direction in the 3D block.

[0043]

3D graph transform

In one embodiment, the transform on the 3D blocks of attributes can use a graph transform. Because our point cloud is partitioned into 3D blocks, we can apply the graph transform on each block.

[0044]

Fig. 4A shows the basic idea behind our graph transform. A graph is formed by connecting adjacent points present in the 3D block. Two points $\mathbf{p}_i$ and $\mathbf{p}_j$ are adjacent if the points are at most one position apart in any dimension. Graph weights w_{ij} are assigned to each connection (graph edge) between points $\mathbf{p}_i$ and $\mathbf{p}_j$.

The weights of each graph edge are inversely proportional to the distance between the two connected points.

[0045]

As shown in Fig. 4B, an adjacency matrix **A** including the weights of the graph edges, from which a graph Laplacian matrix **Q** is determined. The eigenvector matrix of **Q** is used as a transform for the attribute values. After the transform is applied, each connected sub-graph has the equivalent of one DC coefficient, and one or more AC coefficients.

[0046]

In contrast to the modified SA-DCT, which always produces only one DC coefficient, the graph transform method generates one DC coefficient for every disjoint connected set of points in the block, and each DC coefficient has a set of corresponding AC coefficients. In the example of Fig. 4A, the graph is composed of two disjoint sub-graphs, so the resulting graph transform produces two DC coefficients and two corresponding sets of AC coefficients.

[0047]

Preprocessing and coding

Fig. 5 shows the preprocessing and coding method according to embodiments of the invention. The input point cloud 101 acquired by the sensor 103 is preprocessed as described with reference to Fig. 1 to generate the point cloud 140 on a uniform grid. Next, the block partitioning 160, intra prediction 165, and 3D transform 180 are applied. Entropies of transform coefficient magnitudes and sign bits are measured. Then, a quantizer 190 is applied to the transform coefficients. For example, a uniform quantizer can be used to quantize the transform coefficients, with a fixed step size set to determine the amount of compression. The quantized transform coefficients, along with any side information, are then entropy coded 195 for output into a bitstream 501.

[0048]

The steps of the method described herein can be performed in a processor 100 connected to memory and input/output interfaces as known in the art.

[0049]

Decoder

Fig. 6 shows the decoding method according to embodiments of the invention. A bitstream 501 is entropy decoded 601 to produce quantized transform coefficients 602, which are inverse-quantized 603 to produce quantized transform coefficients 604. The quantized transform coefficients are inverse transformed 605 to produce a reconstructed residual block 606. Already-decoded point locations 607 can be used to determine the locations of present and missing elements 608 in the set of quantized transform coefficients or in the reconstructed residual block. Using previously-decoded blocks from memory 610, a predictor 611 computes a prediction block 612. The reconstructed residual block is combined or added 609 to the prediction block to form a reconstructed block 613. Reconstructed blocks are spatially concatenated 614 to previously-decoded reconstructed blocks to produce an array of 3D blocks representing the reconstructed point cloud 615 output by the decoder system 600.

[0050]

Effect of the Invention

The embodiments of the invention extend some of the concepts used to code images and video to compress attributes from unstructured point clouds. Point clouds are preprocessed so the points are arranged on a uniform grid, and then the grid is partitioned into 3D blocks. Unlike image and video processing in which all points in a 2D block correspond to a pixel position, our 3D blocks are not necessarily fully occupied by points. After performing 3D block-based intra prediction, we transform, for example using a 3D shape-adaptive DCT or a graph

transform, and then quantize the resulting data.

CLAIMS

[Claim 1]

A method for compressing a point cloud, wherein the point cloud is composed of a plurality of points in a three-dimensional (3D) space, comprising steps:

acquiring the point cloud with a sensor, wherein each point is associated with a 3D coordinate and at least one attribute;

partitioning the point cloud into an array of 3D blocks of elements, wherein some of the elements in the 3D blocks have missing points;

predicting, for each 3D block by using multivariate nearest-neighbor interpolation and extrapolation, attribute values for the 3D block by projecting the attribute values of neighboring 3D blocks to the adjacent edge plane of the 3D blocks, and obtaining a 3D prediction residual block indicative of a difference between the predicted attribute values for the 3D block and the acquired attribute values, associated with each point of the point cloud, in the 3D block;

applying a 3D transform to each 3D prediction residual block using locations of occupied elements to produce transform coefficients, wherein the transform coefficients have a magnitude and sign;

entropy encoding the transform coefficients according the magnitudes and sign bits to produce a bitstream;

wherein the 3D transform is a shape-adaptive discrete cosine transform (SA-DCT) designed for 3D point cloud attribute compression

defining a contour of points as a region, wherein the region encompasses non-empty positions; and

shifting the points in the regions along each direction toward a border of the block so that there are no empty positions in the block along that border, wherein

the steps are performed in a processor.

[Claim 2]

The method of claim 1, further comprising:

converting the point cloud to an octree of voxels arranged on a grid that is uniform, and wherein the partitioning is repeated until the voxels have a minimal predefined resolution.

[Claim 3]

The method of claim 1, wherein each leaf node in the octree corresponds to a point output by the partitioning, and the position of the point is set to a geometric center of the leaf node, and the attribute value associated with the point is set to an average attribute value of one or more points in the leaf node.

[Claim 4]

The method of claim 1, wherein the partitioning is according to a block edge size.

[Claim 5]

The method of claim 1, wherein the prediction for a current block is from the points contained in non-empty adjacent blocks.

[Claim 6]

The method of claim 5, wherein the prediction selects a prediction direction that yields a least distortion.

[Claim 7]

The method of claim 1, wherein the 3D transform applies a graph transform to each block.

[Claim 8]

The method of claim 7, wherein the graph transform produces two DC coefficients and two corresponding sets of AC coefficients.

[Claim 9]

The method of claim 1, wherein each point of the point cloud is associated with at least one attribute.

[Claim 10]

The method of claim 9, wherein the attribute is color information.

[Claim 11]

The method of claim 9, wherein the attribute is reflectivity information.

[Claim 12]

The method of claim 9, wherein the attribute is a normal vector.

[Claim 13]

The method of claim 1, further comprising:
entropy decoding the bitstream to obtain transform coefficients and point locations;
applying an inverse 3D transform to the transform coefficients to produce a 3D prediction residual block;

arranging the elements in the 3D prediction residual block according to the point locations of occupied elements;

predicting, for each 3D prediction residual block, attribute values for the 3D block based on the attribute values of neighboring 3D blocks, and obtaining a 3D prediction block;

combining the 3D prediction block to the 3D prediction residual block to obtain a 3D reconstructed block;

concatenating the 3D reconstructed block to previously-reconstructed 3D blocks to form an array of 3D reconstructed blocks; and

outputting the array of 3D reconstructed blocks as a reconstructed 3D point cloud.

[Claim 14]

The method of claim 13, wherein the arranging of elements according to the locations of the occupied elements is performed before the inverse 3D transform is applied.

[Claim 15]

The method of claim 1, wherein all missing elements in a 3D block are replaced with predetermined values, and wherein all transforms applied in same direction during the shape-adaptive discrete cosine transform process have same lengths, equal to a number of missing and non-missing elements in the 3D block along that direction.