patchDPCC: A Patchwise Deep Compression Framework for Dynamic Point Clouds

Zirui Pan[1], Mengbai Xiao[1*], Xu Han[1], Dongxiao Yu[1], Guanghui Zhang[1], Yao Liu[2]

[1]School of Computer Science and Technology, Shandong University, Qingdao, China
[2]Rutgers University

Appendix

Algorithms of adjusting a P-patch

In this section, we describe two algorithms adjusting point number of a P-patch $\mathbf{P}^P$ to align with a transformed I-patch $\mathbf{P}^{I_t}$. In the cases of the P-patch having less or more points than the transformed I-patch, we need copy points to or delete points from $\mathbf{P}^P$. As described in the methodology section, we fuse two patches and build an octree over their points before the copy or deletion. Suppose that we have m leaf nodes in the octree, and each is a point set

$$\mathbf{T}_i = \mathbf{S}_i^I \cup \mathbf{S}_i^P, i = 1, \ldots, m,$$

where $\mathbf{S}_i^I$ and $\mathbf{S}_i^P$ are sets of points belonging to the transformed I-patch and the P-patch, respectively. We then follow Algorithm 1 to copy points from the transformed I-patch to the P-patch. It is worth noting that we use three priority queues to ensure that the copy operations are not consecutively performed in neighboring leaf nodes.

Algorithm 1 Find points in a transformed I-patch to copy.

Input: $\mathbf{P}^{I_t}$, $\mathbf{P}^P$, m, r, $\{(\mathbf{T}_i, \mathbf{S}_i^I, \mathbf{S}_i^P) : i = 1, \ldots, m\}$, nearest_pts$(\cdot)$ returns the nearest point pair from two point sets, dist$(\cdot)$ returns the distance of two octree nodes.

Initialize Q_0, Q_1, Q_2 ▷ Q_* are priority queues
$\mathbf{U} \leftarrow \emptyset, \mathbf{R} \leftarrow \emptyset$ ▷ $\mathbf{R}$ stores the points to be copied
$s^I \leftarrow 0, s^P \leftarrow 0$
for $i \leftarrow 1$ to m **do**
  if $|\mathbf{S}_i^I| > 0$ and $|\mathbf{S}_i^P| > 0$ **then**
    $s^I \leftarrow s^I + |\mathbf{S}_i^I|, s^P \leftarrow s^P + |\mathbf{S}_i^P|$
    push$(Q_0, (\mathbf{T}_i, \mathbf{S}_i^I, \mathbf{S}_i^P))$
  else if $|\mathbf{S}_i^I| > 0$ **then**
    $\mathbf{U} \leftarrow \mathbf{U} \cup \mathbf{S}_i^I$
  end if
end for
if $s^I - s^P > |\mathbf{P}^{I_t}| - |\mathbf{P}^P|$ **then**
  for $i \leftarrow 0$ to 2 **do**
    while $(\mathbf{T}, \mathbf{S}^I, \mathbf{S}^P) \leftarrow$ pop(Q_i) **do**
      ▷ Pop the element with the most points in $\mathbf{S}^I$
      $p_{min}^I, p_{min}^P \leftarrow$ nearest_pts$(\mathbf{S}^I, \mathbf{S}^P)$
      $\mathbf{R} \leftarrow \mathbf{R} \cup \{p_{min}^I\}$
      if $|\mathbf{R}| \geq |\mathbf{P}^{I_t}| - |\mathbf{P}^P|$ **then**
        return $\mathbf{R}$
      end if
      $\mathbf{S}^I \leftarrow \mathbf{S}^I - \{p_{min}^I\}$
      $\mathbf{S}^P \leftarrow \mathbf{S}^P \cup \{p_{min}^I\}$
      push$(Q_2, (\mathbf{T}, \mathbf{S}^I, \mathbf{S}^P))$
      if $i < 2$ **then**
        for $(\mathbf{T}', \mathbf{S}'^I, \mathbf{S}'^P) \in Q_i$ **do**
          if dist$(\mathbf{T}, \mathbf{T}') < 2r$ **then**
            delete$(Q_i, (\mathbf{T}', \mathbf{S}'^I, \mathbf{S}'^P))$
            push$(Q_{i+1}, (\mathbf{T}', \mathbf{S}'^I, \mathbf{S}'^P))$
          end if
        end for
      end if
    end while
  end for
else
  for $(\mathbf{T}, \mathbf{S}^I, \mathbf{S}^P) \in Q_0$ **do**
    $\mathbf{R} \leftarrow \mathbf{R} \cup \mathbf{S}^I$
  end for
  while $|\mathbf{R}| < |\mathbf{P}^{I_t}| - |\mathbf{P}^P|$ **do**
    $p^U, p^P \leftarrow$ nearest_pts$(\mathbf{U}, \mathbf{P}^P)$
    $\mathbf{R} \leftarrow \mathbf{R} \cup \{p^U\}, \mathbf{U} \leftarrow \mathbf{U} - \{p^U\}$
  end while
  return $\mathbf{R}$
end if

Similarly, Algorithm 2 is designed to delete excessive points in the P-patch.

Each time a point has been copied between two patches, we want to search the neighboring leaf nodes within 2r and push them to the queue with a lower priority. This is achieved by employing the radius search of KD-tree, whose time complexity is $\mathcal{O}(logN)$. N is the point number of a patch. As most N points are copied from the transformed I-patch, the complexity of the Algorithm 1 is $\mathcal{O}(NlogN)$. As for Algorithm 2, searching the neighbors is also the bottleneck, and its time complexity is $\mathcal{O}(NlogN)$ as well.

Algorithm 2 Find points in a P-patch to delete.

Input: $\mathbf{P}^{I_t}$, $\mathbf{P}^P$, m, r, $\{(\mathbf{T}_i, \mathbf{S}_i^P) : i = 1, \ldots, m\}$, randpick$(\cdot)$ randomly picks a point from a point set, dist$(\cdot)$ returns the distance of two octree nodes.

*Corresponding author.

Initialize Q_0, Q_1, Q_2 ▷ Q_* are priority queues
$\mathbf{R} \leftarrow \emptyset$ ▷ $\mathbf{R}$ stores the points to be deleted
for $i \leftarrow 1$ to $m-$ **do**
  if $|\mathbf{S}_i^P| > 0$ **then**
    push$(Q_0, (\mathbf{T}_i, \mathbf{S}_i^P))$
  end if
end for
for $i \leftarrow 0$ to 2 **do**
  while $(\mathbf{T}, \mathbf{S}^P) \leftarrow$ pop(Q_i) **do**
    ▷ Pop the element with the most points in $\mathbf{S}^P$
    $p^P \leftarrow$ randpick$(\mathbf{S}^P)$
    $\mathbf{R} \leftarrow \mathbf{R} \cup \{p^P\}$
    if $|\mathbf{R}| \geq |\mathbf{P}^P| - |\mathbf{P}^{I_t}|$ **then**
      return $\mathbf{R}$
    end if
    $\mathbf{S}^P \leftarrow \mathbf{S}^P - \{p^P\}$
    push$(Q_2, (\mathbf{T}, \mathbf{S}^P))$
    if $i < 2$ **then**
      for $(\mathbf{T}', \mathbf{S}'^P) \in Q_i$ **do**
        if dist$(\mathbf{T}, \mathbf{T}') < 2r$ **then**
          delete$(Q_i, (\mathbf{T}', \mathbf{S}'^P))$
          push$(Q_{i+1}, (\mathbf{T}', \mathbf{S}'^P))$
        end if
      end for
    end if
  end while
end for

Visualization results

The visualization results are shown in Figure 1. Our approach clearly maintains the geometric details in the original point clouds and reconstructs visually appealing results.

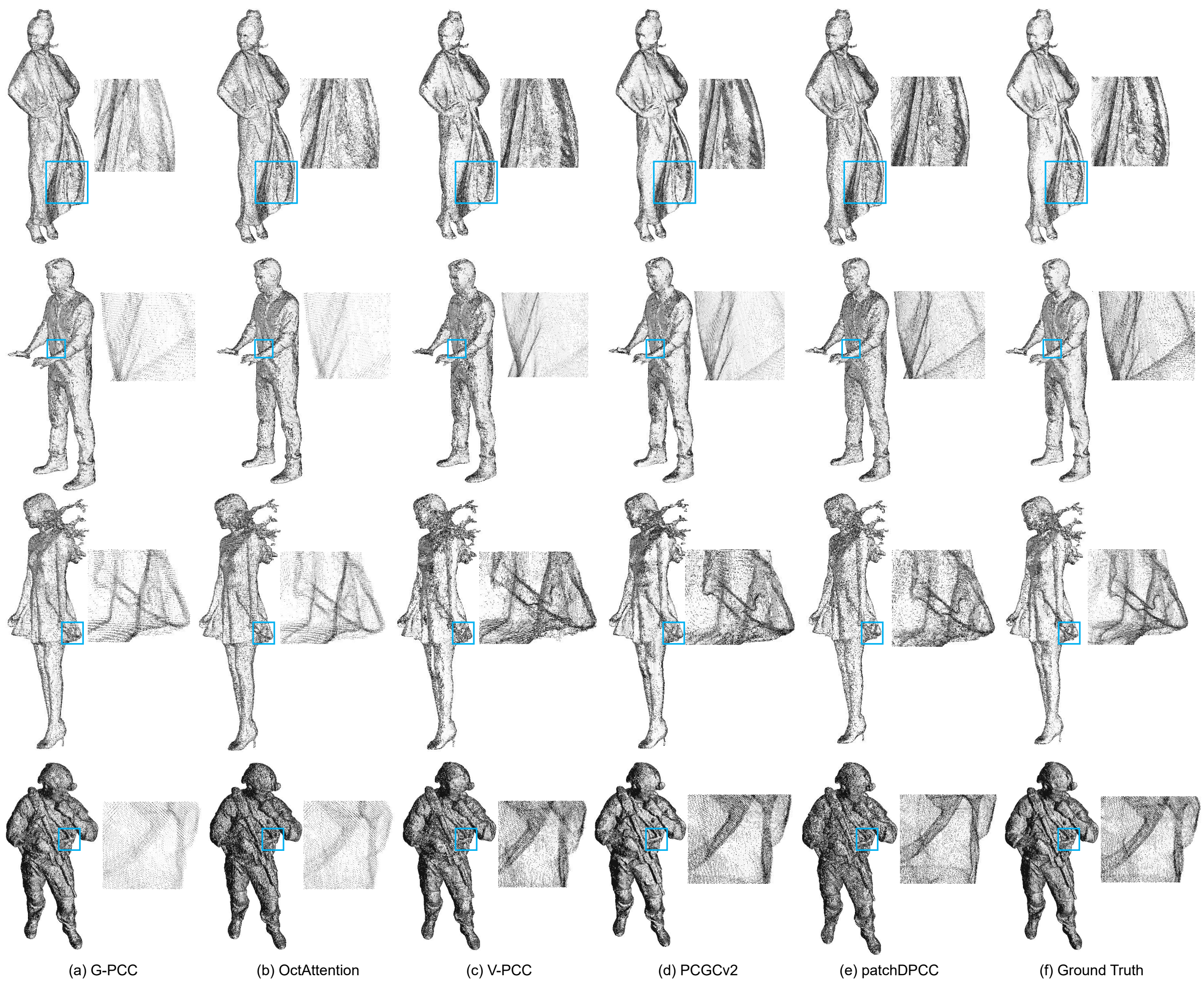

Figure 1: Comparing visual results on *MPEG 8i* dataset using different methods(a-e), and the ground truth(f)