Supplementary Materials and Methods

H-1PV capsid 3D model

The application described here calculates the hypothetical center of mass of the virus from the capsid PDB file. The application computes the distance between the center of mass and each of the atoms forming the capsid. Then each "Center of mass / atom" distance is uploaded into the original PDB file, replacing the b-factor field that can finally be depicted as a colored gradient with the PyMol program. The listing of this application is displayed below. It was written in C++ using the Trolltech Qt framework. It includes two types of objects: "Molecule" (defined in POINT 2 and implemented in POINT 3) representing the whole viral capsid and being a composition of "Atom" objects (defined in POINT 4 and implemented in POINT 5), driven by a main function (described in POINT 1).

1. **Main function that drives the program –**

```
#include "atom.h"
#include "molecule.h"

#include <QApplication>
#include <QFile>
#include <QTextStream>
#include <QString>
#include <QMessageBox>
#include <iostream>
   using namespace std;

int main(int argc, char *argv[])
{
   QApplication app( argc, argv );
   QFile file( "h1.pdb" );
   if ( !file.open( QIODevice::ReadOnly | QIODevice::Text ) )
   {
      QMessageBox::information( 0, "CM", "File not found" );
   }

   QTextStream in( &file );
   Molecule capside;
   while ( !in.atEnd() )
   {
      QString line = in.readLine();
      Atom at( line );
      if ( at.isValid() )
         capside.addAtom( at );
   }

   file.close();

   capside.centerOfMass();

   file.open( QIODevice::ReadOnly | QIODevice::Text );

   QFile outFile( "h12.pdb" );
   if ( !outFile.open( QIODevice::WriteOnly | QIODevice::Text ) )
   {
      QMessageBox::information( 0, "CM", "Can not open output file" );
   }

   QTextStream out( &outFile );
   while ( !in.atEnd() )
   {
      QString line = in.readLine();
      Atom at( line );
      if ( at.isValid() )
      {
         at.setDistance( capside.getCMx(), capside.getCMy(), capside.getCMz() );
         out << at.newString( line ) << "\n";
      }
      else
         out << line << "\n";
   }

   file.close();
   outFile.close();

   QMessageBox::information( 0, "CM", "Finished" );

   app.exec();
}
```

2. **Header file for the "Molecule" object -** This file contains the interface of the "Molecule" object, which is basically a QList of "Atom" objects having the capacity to calculate its own center of mass through the centerOfMass() public member.

```
#ifndef MOLECULE_H
#define MOLECULE_H

#include <QList>
#include "atom.h"

class Molecule
{
public:
   Molecule();
   void addAtom( Atom a );
   void centerOfMass();
   float getCMx() { return CMx; }
   float getCMy() { return CMy; }
   float getCMz() { return CMz; }

private:
   QList< Atom > list;
   float CMx;
   float CMy;
   float CMz;
   float totalMass;
};

#endif
```

3. **Implementation of the "Molecule" object -** This file contains the implementation of the two public members: (i) addAtom() allowing to add a newly identified atom from the input pdb into the QList; and (ii) centerOfMass() calculating the x, y, z positions of the center of mass of the capsid (geometric center) once the QList is fully loaded.

```
#include "molecule.h"
#include "atom.h"

#include <QLocale>
#include <QMessageBox>

Molecule::Molecule()
{

}

void Molecule::addAtom( Atom a )
{
   list.append( a );
}

void Molecule::centerOfMass()
{
   totalMass = 0.00;
   CMx = 0.00;
   CMy = 0.00;
   CMz = 0.00;

   for ( int i = 0; i < list.size(); i++ )
   {
      Atom a = list.at( i );
      totalMass += a.getMass();
   }

   for ( int i = 0; i < list.size(); i++ )
   {
      Atom a = list.at ( i );
      CMx += a.getMass() * a.getX() / totalMass;
      CMy += a.getMass() * a.getY() / totalMass;
      CMz += a.getMass() * a.getZ() / totalMass;
   }

      const QLocale l;
      QString s = l.toString( CMx );
      QMessageBox::information( 0, "CMx", s );
      s = l.toString( CMy );
      QMessageBox::information( 0, "CMy", s );
      s = l.toString( CMz );
      QMessageBox::information( 0, "CMz", s );
}
```

4. **Header file for the "Atom" object** - This file contains the interface of the "Atom" object, which is storing the x, y, z position of each atom, its mass and its distance from the center of mass of the whole capsid.

```
#ifndef ATOM_H
#define ATOM_H

class QString;

class Atom
{
public:
   Atom( QString s );
   bool isValid();
   void setDistance( float CMx, float CMy, float CMz );
   float getMass() { return mass; }
   float getX() { return x; }
   float getY() { return y; }
   float getZ() { return z; }
   float getDistance() { return d; }
   QString newString( QString s );

private:
   float x;
   float y;
   float z;
   float mass;
   float d; // Distance to the center of mass of the molecule
   bool valid;

};

#endif
```

5. **Implementation of the "Atom" object -** This file contains the implementation of the following public members: (i) Atom(QString s) is the constructor of the Atom object that takes a line of the input pdb file as a parameter and loads x, y, z, mass values using a regular expression to extract the appropriate information, (ii) newString(QStrings) that creates the line for this atom that will be written down in the output pdb file, with the newly calculated b-factor field value replaced by the distance in between this atom and the center of mass of the capsid, and (iii) setDistance(float CMx, float CMy, float CMz) that uses the Pythagorean theorem and computes the center of mass/atom distance d on the basis of the x, y, z positions of the atom and the CMx, CMy, CMz positions of the center of mass of the capsid.

```
#include "atom.h"

#include <QRegExp>
#include <QRegExpValidator>
#include <QMessageBox>
#include <QLocale>
#include <iostream>
   using namespace std;
#include <math.h>

Atom::Atom( QString s )
{
   QRegExp rx( "((ATOM|HET)\\s+\\d+\\s+\\s+([\\w\\*]+).+\\s+) \
                (-?\\d+\\.\\d+)(\\s+)?(-?\\d+\\.\\d+)(\\s+)? \
                (-?\\d+\\.\\d+)(\\s+1\\.00\\s+)(\\d+\\.\\d+)(.+)" );
   QRegExpValidator v( rx, 0 );
   int pos = 0;

   if ( v.validate( s, pos ) == QValidator::Acceptable )
   {
      valid = TRUE;
      int p = rx.indexIn( s );
      if ( p >= -1 )
      {
         x = rx.cap( 4 ).toFloat();
         y = rx.cap( 6 ).toFloat();
         z = rx.cap( 8 ).toFloat();

         mass = 1.00;
         if ( rx.cap( 3 ).at( 0 ) == QChar( 'C' ) )
            mass = 12.00;
         if ( rx.cap( 3 ).at( 0 ) == QChar( 'N' ) )
            mass =  14.00;
      }
   }
   else
      valid = FALSE;
}

bool Atom::isValid()
{
   if ( valid == TRUE )
      return TRUE;
   else
      return FALSE;
}
```

```
QString Atom::newString( QString s )
{
   QRegExp rx( "((ATOM|HET)\\s+\\d+\\s+\\s+([\\w\\*]+).+\\s+) \
                (-?\\d+\\.\\d+)(\\s+)?(-?\\d+\\.\\d+)(\\s+)? \
                (-?\\d+\\.\\d+)(\\s+1\\.00\\s+)(\\d+\\.\\d+)(.+)" );
   QRegExpValidator v( rx, 0 );
   int pos = 0;

   if ( v.validate( s, pos ) == QValidator::Acceptable )
   {
      valid = TRUE;
      int p = rx.indexIn( s );
      if ( p >= -1 )
      {
         const QLocale l;
         QString b = l.toString( d, 'f', 3 );

         QString out;
         out += rx.cap( 1 );
         out += rx.cap( 4 );
         out += rx.cap( 5 );
         out += rx.cap( 6 );
         out += rx.cap( 7 );
         out += rx.cap( 8 );
         out += rx.cap( 9 );
         out += b;
         out += rx.cap( 11 );

//       QMessageBox::information( 0, "CM", out );

         return out;
      }
   }
   else
      valid = FALSE;

   return QString( "" );
}

void Atom::setDistance( float CMx, float CMy, float CMz )
{
   float dist;
   dist = sqrt( pow( getX() - CMx, 2.0 ) + pow( getY() - CMy, 2.0 ) + \
          pow( getZ() - CMz, 2.0 ) );
   d = dist;
}
```