Tudományos célú képalkotás a POV-Ray segítségével

Egy kis munkával a Persistence of Vision Raytracer (POV-Ray) felhasználható arra, hogy lebegőpontos tudományos adatfájlok alapján lélegzetelállító háromdimenziós ábrákat hozzunk létre.

Meteorológus vagyok a ***Michigani Központi Egyetemen***, ahol az ***Illinois Egyetemen*** dolgozó kollégáimmal a szupercellás zivatarok viselkedését kutatom. Ezek a hosszú életű örvénylő szörnyetegek minden tavasszal nagy felfordulást okoznak a ***Nagy Síkságokon*** az ***Egyesült Államokban***. E félelmetes viharok tanulmányozásának elsődleges eszköze számomra egy ***NCOMMAS*** nevű numerikus modell, egy ***FORTRAN 90*** nyelven írt számítógépes program, amely a fizikai egyenletek felhasználásával emulálja a légkör háromdimenziós időbeli állapotváltozását. Ez a modell egy négy órás vihar szimulációja során roppant mennyiségű adatot állít elő, amely még veszteséges tömörítéssel is 200 GB nagyságrendű marad. A kutatásaim során felmerült egyik nagy kihívás az volt, hogy módot találjak ezeknek az adatoknak olyan tudományos igényű megjelenítésére, amellyel betekintést nyerhetnénk a szimulált vihar fizikai természetébe.

A háromdimenziós adatok megjelenítésének egyik módja egy sugárkövető (***raytracer***) használata. A sugárkövető egy olyan számítógépes program, amely a pontkép előállításához a fény viselkedését szimulálja, amint az a háromdimenziós térben elhelyezett virtuális tárgyakkal kölcsönhatásba lép (*1. ábra*). Az így kapott bittérkép megjeleníthető ezután a számítógép képernyőjén, vagy lemezre menthető valamilyen ismert formátumban, mint amilyen a ***PPM*** vagy a ***TIFF***. A ***Persistence of Vision Raytracer***, vagy röviden ***POV-Ray*** egy népszerű nyílt forráskódú sugárkövető program, amely akkor keltette fel az érdeklődésemet, amikor a 90-es évek közepén a ***Wisconsin Egyetemen*** a doktori disszertációmat írtam. Abban az időben konvektív leáramlások – a viharfelhőkben időnként kialakuló lefelé irányuló légáramlások – háromdimenziós adatainak megjelenítésére kerestem megfelelő programot. Mivel a tudományos életben hozzá voltam szokva a megosztott forráskódokhoz és végzős diákként anyagilag sem álltam jól, olyan ingyenes és forráskódban terjesztett programot kerestem, amit letölthetek és különleges igényeimhez igazíthatok. A logikus választás akkor a ***POV-Ray*** volt és ma is megfelel az igényeimnek, amikor a kutatásaim adatait sugárkövetési módszerrel előállított képen szeretném megjeleníteni.

A tudományos adatok leképezése nem az a feladat, amelyre a ***POV-Ray*** eredetileg készült, és kevés kutató használja a programot erre a célra. Más sugárkövető programcsomagok a numerikus modellek terén jobban támogatják a kutatók munkáját, de ezek zárt forráskódúak és meglehetősen drágák. Ebben a cikkben körvonalazom azt a módszert, amellyel a ***POV-Ray*** képessé tehető a háromdimenziós tudományos adatok szintfelületeinek közvetlen megjelenítésére.

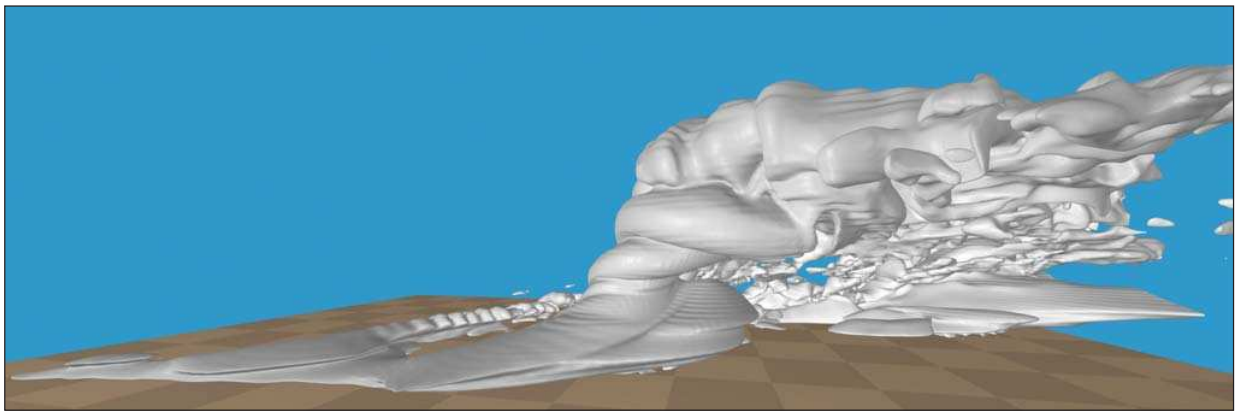
A forráskód megszerzése

Bár a ***POV-Ray*** bináris változatban is elérhető ***Linux***, ***Mac OS*** és ***Microsoft Windows*** operációs rendszerekhez, nekünk szükségünk lesz a forráskódra is ahhoz, hogy a foltokat alkalmazni tudjuk, és további módosításokat hajthassunk végre. A cikkben az írás idején elérhető legfrissebb, 3.5 változatot használom. A ***POV-Ray*** letöltési oldalán választanunk kell a Unix, Linux és általános forráskód közül, és meg kell szereznünk ***Ryouichi Suzuki Density File*** kiterjesztés-foltját (lásd a hálózaton elérhető címek között).

Ez egy ***Zip*** fájl, amely a POV-Ray fájljainak egy részét helyettesítő forráskódot tartalmaz. A ***pov35dfjs.zip*** fájlt a ***povray-3.50c/src*** könyvtárban kell kicsomagolni, ahol felül fog írni tizenhárom fájlt.

A jelenetek és a szintfelületek

A ***POV-Ray*** olyan jelenetleíró fájlokkal dolgozik, amelyek minden információt tartalmaznak a bittérképes kép létrehozásához. A ***POV-Ray*** saját, a honlapján jól dokumentált jelenetleíró nyelvet használ. Ha korábban nem használtunk semmilyen sugárkövető programot, azt javaslom, hogy ismerkedjünk meg a módszer alapjaival és a ***POV-Ray*** jelenetleíró fájljaival, még mielőtt a forráskódhoz hozzányúlnánk. A ***POV-Ray***-ben leképezett elemeket objektumoknak nevezzük, amelyekre példa a téglatest, gömb, tórusz vagy a sík. A felhasználó megadja az objektum jelenetben elfoglalt helyét, a méreteit, színeit, megvilágítási jellemzőkét és így tovább. Ezek az adatok egy jelenetleíró fájlban szerepelnek, amelyeket a ***.pov*** kiterjesztésük miatt esetenként ***pov***-fájloknak is neveznek. A jelenetleíró fájlok közönséges szövegfájlok, amelyek értelmezését a ***POV-Ray*** futásidőben végzi el.



1. ábra Egy teljes szupercellás vihar légi felvétele körülbelül 30 kilométeres távolságból a POV-Ray segítségével leképezve

Egy általánosan használható objektum a szintfelület. Ez egy olyan háromdimenziós forma, amelynek a felülete az azonos függvényértékkel rendelkező pontok megjelenítésével áll elő. A függvény állandó értékét a felhasználó választhatja ki. A ***POV-Ray*** számos olyan előre definiált objektumot tartalmaz, amely valójában szintfelületnek is tekinthető. A következő jelenetleíró fájlból származó részlet például egy 0,7 egység sugarú, szürke színű gömböt képez le, melynek középpontja az origóban, vagyis a `(0,0,0)` Descartes-koordinátákkal jellemzett pontban helyezkedik el:

```
sphere
{
    <0,0,0>, 0.7
    pigment { rgb .5}
}
```

Ugyanez az objektum szintfelületként is megadható lenne az alábbi módon:

```
#declare R = 0.7
isosurface
{
    function { x*x + y*y + z*z - R*R}
    pigment { rgb .5}
}
```

A meghatározás hátterében az áll, hogy az R sugarú gömb az alábbi matematikai egyenlettel írható le:

$$x^2 + y^2 + z^2 - R^2 = 0$$

A szintfelületeknek ez a sokoldalúsága volt az a tulajdonság, ami miatt ezt az objektumot választottam a zivatarok ábrázolásához.

Sűrűségfájlok

A gömb példájában egy matematikai függvényt használtam a szintfelület értékének kiszámításához. A zivatarjaimhoz használt numerikus modell adatai nem írhatók fel egyetlen matematikai függvényként, ehelyett egy olyan háromdimenziós lebegőpontos tömbökről van szó, amelyek minden rácspontban valamilyen modellváltozót tartalmaznak, ami például lehet hőmérséklet, szélsebesség vagy felhősűrűség (*2. ábra*).

A ***POV-Ray 3.5*** változatának van egy sűrűségfájl nevű szolgáltatása, amely lehetővé teszi a függvények rácspont-értékekként való leképezését. A ***POV-Ray*** dokumentációja a következőképpen írja le a sűrűségfájlokat: „A sűrűségfájl-mintázat egy olyan háromdimenziós bittérkép-mintázat, amely egy ***<0,0,0>*** és ***<1,1,1>*** koordinátájú pontok között elhelyezkedő egységnyi kockát tölt be. Az adatfájl egy a ***POV-Ray*** számára készült, ***df3*** nevű nyers bináris fájlformátum."

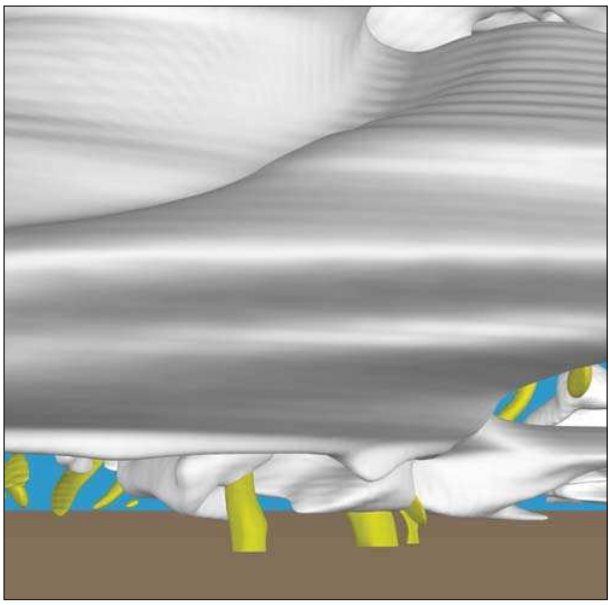
A sűrűségfájlok olyan függvényként használhatóak, amelyeket a szintfelület-objektum paraméterként kap meg. Íme egy példa a szintfelület leképezéséhez használt sűrűségfájlra:

```
#declare DENSFUNC=function
{
    pattern
    {
        density_file df3 "cloud.df3"
        interpolate 1
    }
}
isosurface { function {  0.1 - DENSFUNC(x,y,z) }
```

A fenti példában a ***cloud.df3*** fájl segítségével egy 0,1 értékű szintfelület állítanánk elő egy háromvonalas (trilineáris) interpolációs séma használatával (az interpolációról rövidesen lesz még szó).

A sűrűségfájl formátuma szigorúan kötött, az adatokat 8 bites értékek képviselik (0 és 255 közti előjel nélküli egészek), amelyeket a program alakít át 0,0 és 1,0 közti értékekre. Mivel az én zivatar-adataim 32 bites lebegőpontos értékek, a sűrűségfájl-formátum nem alkalmas közvetlenül az eredeti ***POV-Ray*** 3.5 változattal való használatra.

Itt lép be ***Ryouichi Suzuki***, aki 1996 óta fejleszt a ***POV-Ray*** számára kiegészítő kódokat. Ő készítette a ***POV-Ray*** 3.0-hoz azokat a foltokat is, amelyek elsőként vezették be a 3.5 változatba már beépített objektumként szereplő szintfelület-objektumokat. ***Suzuki*** fent említett ***zip***-fájljában lévő kód tartalmaz olyan eljárásokat is, amelyek kiterjesztik a ***POV-***



2. ábra Egy példa többszörös szintfelületekre, ahol a központban a szupercella felhőfalnak nevezett tartománya áll. A felhőfal alatt az előtérben látható sárga szintfelületek a tornádószerű örvénylő mozgást mutatják.

Ray sűrűségfájljainak lehetőségeit, képessé téve a programot többek közt a lebegőpontos sűrűségfájlok alapján történő leképezésre is.

Amikor sűrűségfájlokat használunk függvények helyett, felmerülhet valakiben a gondolat, hogy míg a függvények folytonos kifejezések – vagyis az ***x***, ***y*** és ***z*** térkoordinátákhoz bármilyen lebegőpontos érték tartozhat – a sűrűségfájl az adatoknak olyan diszkrét halmaza, amelyet a tömbök egész indexei jellemeznek. Egy kép leképezésénél a rácspontok közti területet interpoláció segítségével kell kitölteni. A két rendelkezésre álló interpolációs eljárás a háromvonalas interpoláció és a háromirányú köbös simulógörbe (tricubic spline). A háromvonalas interpoláció gyorsabb, de gyakran nem ad olyan egyenletes eredményt, mint a simulógörbe.

A modelladatok betöltése a POV-Ray-be

Miután a ***POV-Ray 3.5***-re feltelepítettük ***Suzuki*** sűrűségfájlra vonatkozó foltjait, a rendszer készen áll arra, hogy lebegőpontos értékek alapján képezzen le szintfelületeket – amennyiben az adatok ***df3*** formátumban, vagy ***Suzuki*** kiterjesztett formátumában rendelkezésre állnak. Az én esetemben az adatok több száz gigabájtnyi *HDF*-fájlként (hierarchical data format, hierarchikus adatformátum) tárolódnak, amely formátum kifejezetten numerikus modellek adatainak a tárolására lett kifejlesztve. Mivel engem nem csak néhány szintfelületes kép előállítása érdekelt, hanem szerettem volna az előállított több száz, esetleg több ezer képből animációt is létrehozni, a ***HDF-df3*** átalakítás nem jöhetett szóba. Ehelyett inkább vizsgálni kezdtem a ***POV-Ray*** eljárásait, amelyekkel a sűrűségfájlokat kezeli és reménykedtem, hogy sikerül a kódot úgy módosítanom, hogy olvasni tudja a *HDF*-fájljaimat.

Fontos volt számomra az is, hogy a módosításaim ne csökkentsék az eredeti program használhatóságát és teljesen kompatibilis maradjon a folt nélküli változattal. A célomat úgy értem el, hogy a jelenetleíró fájlomhoz olyan új objektumokat adtam, amelyeket a módosított változat képes volt elemezni és leképezni, míg az összes többi objektum változatlan maradt. Az általam módosított kód központi része a ***pattern.cpp*** fájlban található, amely a `Read_Density_File` eljárást tartalmazza. Ahogy a neve is sejteti, ez az eljárás olvassa be a sűrűségfájlt egy háromdimenziós tömbbe. Ennek az eljárásnak a mintájára létrehoztam egy új eljárást `Read_Hdf_File` néven, amely az én leírófájlomat olvassa be a ***POV-Ray***-be. Ez az a rész, ahol a legtöbb módosítást igényli a programkód, ha ragaszkodunk a saját adatformátumunkhoz. Az *1. listán* a `Read_Hdf_File` eljárás rövidített változatát láthatjuk.

A `Read_Hdf_File` függvény olvassa be a *HDF* lebegőpontos adatait a `mapF` nevű háromdimenziós lebegőpontos tömbbe, amely ezután már sűrűségfájlként kezelhető. A ***hystory.c*** fájlban külön írtam meg azokat a kódokat, amelyekre a *HDF* I/O-eljárásai hivatkoznak a ***pattern.cpp*** fájlban. A saját adatfájl-formátumunkhoz saját magunknak kell megírnunk azt a formátumra jellemző kódot, amely a háromdimenziós adatainkat beolvassa a ***POV-Ray***-be.

Módosítanom kellett még néhány fájlt annak érdekében, hogy a ***POV-Ray*** felismerje a *HDF* fájlformátumot és hogy lehetővé váljon képenként egynél több modellváltozó leképezése. Az *1. táblázat* sorolja fel a módosított fájlokat és rövid leírást ad az elvégzett változtatásokról.

A *HDF* fájl a sűrűségfájllal ellentétben lehetővé tesz fájlonként egynél több változó tárolását is. Az esetemben minden egyes *HDF* fájl a modell egy bizonyos időpontbeli állapotát írja le, és fájlonként 12 háromdimenziós változót tartalmaz. Gyakran sokkal látványosabb, ha egy képen több változót (felhő, eső, jégeső, hó) együtt is megfigyelhetünk. Ennek megvalósítására a *HDF* fájlformátum ábrázolásához egy új tokent, a ***HDF_TOKEN***-t hoztam létre (szemben az eredeti df3 ábrázolását végző DF3_TOKEN nevű tokennel), és egy Var nevű új karaktertömböt hoztam létre a `Density_file_Data_Struct` szerkezetben. A Var a jelenetleíró fájlban kap értéket, és a *HDF*-eljárásoknak paraméterként átadva meghatározza, hogy melyik modellváltozó legyen kiválasztva. A karakterláncként tárolt változónév értelmezéséhez egy további `case` utasítást adtam a ***parstxtr.cpp*** fájl `Parse_PatternFunction` függvényéhez (*2. lista*). Láthatjuk a `Parse_Comma` és `Parse_C_String` hozzáadását is, amelyek a beolvasandó változót csípik el.

A jelenetleíró fájl

Most már minden részlet a helyén van ahhoz, hogy létrehozzunk egy ***POV-Ray*** által értelmezhető jelenetleíró fájlt. Ehhez a ***Suzuki***-féle sűrűségfájl-kiterjesztés honlapján megtalálható példafájlt használtam sablonként egy kicsit az igényeimhez igazítva. A *3. lista* tartalmazza a teljes jelenetleíró fájlt, amit az 1. ábrán látható kék hátterű és burkolófelülettel ellátott felhősűrűség-szintfelület leképezéséhez használtam. A tetejétől indulva először a #version utasítást látjuk, ez teszi lehetővé a nem hivatalos ***POV-Ray*** változatom működését. Az ezután következő kilenc `#declare` utasítás a szintfelületet tartalmazó téglatestet és a tengelyek beosztását határozza meg.

1. lista A Read_Hdf_File listájának rövidített változata a pattern.cpp fájlból

```
void Read_Hdf_File (DENSITY_FILE * df)
{
 Locate_Density_File(df->Data->Name);
 df->Data->Type = 1; //floating point data
 Open_HDF_File(df->Data->Name);
 //povray needs array dimensions
 Read_HDF_File_Geometry(nx,ny,nz);
 df->Data->Sx = nx;
 df->Data->Sy = ny;
 df->Data->Sz = nz;
 //this array will contain density file data
 Allocate_Memory(mapF,nx,ny,nz);
 //read variable into mapF array
 Get_HDF_File_Data(Var,mapF,nx,ny,nz);
 //density file pointer now points to model data
 df->Data->DensityF = mapF;
}
```

2. lista A HDF_TOKEN case-ágában szükség van egy további elemző részre, amely lehetővé teszi annak megadását, hogy melyik változó leképezése történjen. A kódrészlet a parstxtr.cpp fájl Parse_PatternFunction eljárásában található.

```
EXPECT
  CASE (DF3_TOKEN)
    New->Vals.Density_File->Data->Name =
        Parse_C_String(true);
    Read_Density_File(New->Vals.Density_File);
    EXIT
  END_CASE
  CASE (HDF_TOKEN)
    New->Vals.Density_File->Data->Name =
        Parse_C_String(true);
    Parse_Comma();
    New->Vals.Density_File->Data->Var =
        Parse_C_String(true);
    Read_Hdf_File(New->Vals.Density_File);
    EXIT
  END_CASE
```

Továbbhaladva a jelenetleíró fájlban a színek és a felület kidolgozásának paramétereinek, valamint a kameraállás és megvilágítás jellemzőinek megadása történik. Az ezt követő sorok tartalmazzák a lényeget, a szintfelület meghatározását. A `QCFUNC` egy függvény, amely a forrásadatokat a ***supercell.ck10990.hdf*** nevű ***HDF***-fájlból olvassa be és ezekből a leképezéshez a `QC` változó tartalmát használja fel (amely változó a felhősűrűséget tárolja). Az interpolációhoz a köbös simulógörbét választjuk, és a teljes tartományra érvényesítjük a fokbeosztást, így az összes olyan térbeli koordináta, mint a kameraállás, megvilágítás helyzete, egybeesik az adatértékekkel. Alapértelmezésben a ***POV-Ray*** tartománya mindhárom irányban a 0,0-tól 1,0-ig terjedő tartományt használná.

1. táblázat ***A modelladatok POV-Ray-ben történő alkalmazását biztosító módosítások listája***

Fájl	Módosítás
pattern.cp	A modelladatokat beolvasó Get_HDF_File_Data eljárás hozzáadása.
pattern.h	A Read_Hdf_File meghatározásának hozzáadása.
parstxtr.cpp	A HDF_TOKEN case-blokkjának hozzáadása.
tokenize.cpp	A HDF_TOKEN hozzáadása a Reserved_Words tömbhöz.
frame.h	A char *Var1 hozzáadása a Density_file_Data_Struct szerkezethez.
parse.h	A HDF_TOKEN hozzáadása a TOKEN_IDS-hez..

Létrehoztam egy `QCISOSFC` nevű makrót, amely paraméterként a leképezni kívánt szintfelület értékét és a szintfelület átlátszóságát kapja meg. Az szintfelület átlátszósága egy jól használható tulajdonság, amikor két egymást fedő szintfelületet ábrázolunk. Például érdemes áttetszővé tenni a felhőt, amikor egyidejűleg jégsűrűség-szintfelületet is ábrázolunk, mivel a jég gyakran található a viharfelhők belsejében.
A makróban leképezés szintfelület-függvényeként a feljebb definiált `QCFUNC`-ot választjuk. A kiválasztott szintfelület azokat a felhősűrűség-értékeket veszi figyelembe, amelyek nagyobbak a felülethez kiválasztott 0,0002 értéknél.
A `max_gradient` paraméter lényegében azt határozza meg a ***POV-Ray*** számára, hogy mennyi munkát fordítson a szintfelület meghatározására. A működés szempontjából azt rögzíti a program számára, hogy mi az a maximális gradiens (változási gyorsaság a távolság függvényében), amellyel a függvény egy adott szintfelület-pont környezetében lévő felület-adatokat leírja. Ezt a számot nagyon körültekintően kell megválasztani. Túl kicsire választva az értéket lyukak lesznek a felületben, esetleg egyáltalán nem kapunk felületet; túl nagyra választva viszont a ***POV-Ray*** sokkal hosszabb ideig fog futni, mint az szükséges lenne. Némi gyakorlat kell ahhoz, hogy a megfelelő értéket válasszuk. Én a 0,0002 értéket választottam, ami a felhősűrűség körülbelül 0,0-tól 0,01 értéktartományához viszonyítva első ránézésre kicsinek tűnik. A ***POV-Ray*** egy kép túl nagy vagy túl kicsi értékkel végrehajtott leképezése után javasol egy olyan értéket, amit a leképezés alapján megfelelőnek tart.

Képek és egyebek létrehozása

A program változtatásokkal történő lefordításához néhány kisebb módosítást kell végrehajtanunk az ***src/Makefile*** fájlban, amely a ***POV-Ray*** könyvtárának felső szintjén lévő `configure` első futtatásakor jön létre. Ez az eset akkor áll fenn, ha az adatfájljaink beolvasó eljárásaihoz külső programkönyvtárakat használunk, vagy ha a fájlok I/O műveleteit külön kóddal oldottuk meg.

3. lista A cloud.pov fájl

```
#version unofficial dfe 3.5;
#include "colors.inc"

#declare x0 = 0.0;
#declare x1 = 700.0;
#declare y0 = 0.0;
#declare y1 = 600.0;
#declare z0 = 0.0;
#declare z1 = 80;

#declare scalex = (x1-x0+1);
#declare scaley = (y1-y0+1);
#declare scalez = (z1-z0+1);

#declare R = 0.7;
#declare G = 0.7;
#declare B = 0.7;

#declare AMBIENT    = 0.5;
#declare DIFFUSE    = 1.1;
#declare SPECULAR   = 0.3;
#declare ROUGHNESS  = 0.01;
#declare BRILLIANCE = 1.0;

camera {
    up           <0,0,1>
    sky          <0,0,1>
    right        <3.0,0,0>
    direction    <1.0,0,0>
    location     <420,70,70>
    look_at      <370,300,90>
}

light_source { <100,100,100> color Gray25
shadowless}
light_source { <400,200,30> color Gray20 }
light_source { <1000,-500,150> color Gray25 }
light_source { <-400,-500,150> color Gray25 }

#declare QCFUNC = function {  pattern{
    density_file hdf "supercell.ck10990.hdf","QC"
    interpolate 2 //tricubic spline
    frequency 0
scale <scalex,scaley,scalez> }  }

#macro QCISOSFC(iso,trans)
isosurface{  function{  -QCFUNC(x,y,z) }
threshold -iso
max_gradient 0.0002
contained_by{ box{ <x0,y0,z0>,<x1,y1,z1>} }
texture{  pigment{ color rgbt<R,G,B,trans>}
finish{ ambient AMBIENT diffuse DIFFUSE
    specular SPECULAR roughness ROUGHNESS
    brilliance BRILLIANCE}   } _shadow
}
#end

QCISOSFC(0.0002,0.0) // render cloud

box {  <x0,y0,z0> <x1,y1,z0> // tiles 5km square
pigment { checker color NewTan,
    color .90*NewTan scale 50}
finish { ambient 0.5 diffuse 0.5}  }

background { SkyBlue}  // what else?
```

A program lefordítása után a parancssorból indíthatjuk a *POV-Ray*-t. Az alábbi parancs beolvassa a *cloud.pov* fájlt és egy *600x400* felbontású élsimított *PPM*-fájlt állít elő megjelenítve a képernyőn a leképezés folyamatát:

```
/home/orf/povray-3.50c-orf/src/povray +D \
Input_File_Name=cloud Width=600 Height=400 \
Antialias=on Output_File_Type=P
```

Miután az adataink alapján a *POV-Ray* sikeresen elkészítette a képet, a beállítható eszközök széles választékával alakíthatjuk a leképezést az igényeinknek legmegfelelőbb formára. Ha folyamatosan változó adatokkal rendelkezünk, kézenfekvő igény, hogy ezekből mozgóképet hozzunk létre. Én *Python* parancsfájlokkal oldottam meg a különböző *POV-Ray* példányok indítását a leképezőtelepem egyes processzorain, ahol a processzorok párhuzamosan dolgoznak a modell különböző időállapotainak adatain. A kapott *PPM*-fájlokat ezután az `mjpegtools` segítségével illesztem össze mozgóképpé. A kutatói honlapomról letölthető néhány ilyen animáció. Sztereóképeket és mozgóképeket is létrehoztam az osztályunk *GeoWall* rendszere számára. A sztereó képpárok létrehozása a *POV-Ray* számára nem jelent gondot és valóban új megvilágításba helyezi az adatainkat. A *POV-Ray* használata a modelljeim adatainak megjelenítésére egy sereg új izgalmas lehetőség felé nyitotta meg számomra az utat, s remélem, hogy ebben nem vagyok egyedül.

A cikkhez kapcsolódó hivatkozások a ➲ *www.linuxjournal.com/article/7754* címen érhetőek el.

Linux Journal 2004. november, 127. szám

Leigh Orf a Michigani Központi Egyetem légkörkutatással foglalkozó professzora, és hosszú ideje Linux felhasználó. Tudományos kutatási területei közé tartozik a viharok valósághű szimulációja és megjelenítése, amelyekhez nagy teljesítményű Linux-fürtöket használ. Szabadidejében élvezettel főzi saját sörét, rádióamatőrködik, szaxofonozik vagy indul sátortúrára feleségével, Annie-vel.
A leigh.orf@cmich.edu címen érhető el.