11. Gyakorlat
Példaprogramok VisiQuest-ben

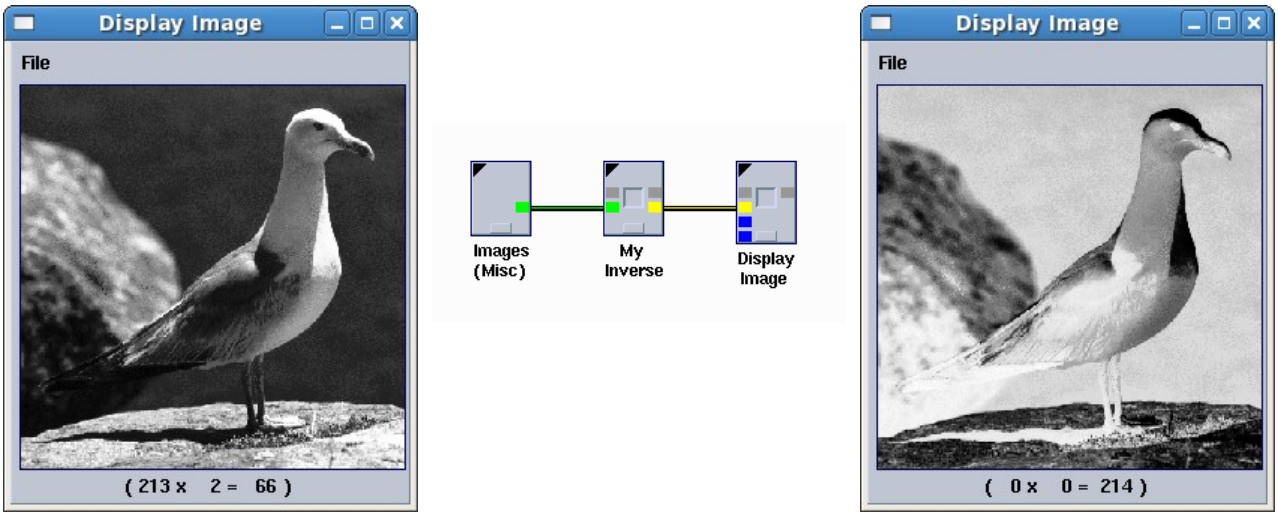
1. Szürkeárnyalatos kép invertálása

A szürkeárnyalatos kép intenzitástartománya 0-255 lesz, ezt várjuk inputként. Az invertált képet úgy állítjuk elő, hogy minden egyes képpont intenzitását kivonjuk 255-ből. Ezalapján a VisiQuest által generált kódot a következőképpen kell módosítani:

```
int run_myinverse(void)
{
   /*-- Put Your Code Here --*/

   /* -main_variable_list */
   char *lib = "myinverse_obj";
   char *rtn = "main";
   kobject in_object = NULL;   /* bemeneti képobjektum */
   kobject out_object = NULL;  /* kimeneti képobjektum */
   unsigned char *plane;       /* memóriaterület az input adat számára */
   unsigned char *res_plane;   /* memóraterület az eredménykép számára */
   int w, h, d, t, e;      /* változók, amelyekben eltároljuk a dimenziókat */
   int cw, ch, ct, pos;    /* segédváltozók */
   /* -main_variable_list_end */

   /* -main_before_lib_call */
   /* bemeneti objektum megnyitása */
   if ((in_object = kpds_open_input_object(clui_info->i_file))
         == KOBJECT_INVALID) {
     kerror(lib, rtn, "Can not open input object %s.\n", clui_info->i_file);
     kexit(KEXIT_FAILURE);
   }

   /* kimeneti objektum megnyitása */
   if ((out_object = kpds_open_output_object(clui_info->o_file))
        == KOBJECT_INVALID) {
     kerror(lib, rtn, "Can not open output object %s.\n", clui_info->o_file);
     kexit(KEXIT_FAILURE);
   }

   /*
      a bemeneti objektum átmásolása a kimeneti objektumba, minden attribútum
      másolódik
   */
   if (!kpds_copy_object(in_object, out_object)) {
     kerror(lib, rtn, "Can not copy input object to output object.\n");
     kexit(KEXIT_FAILURE);
   }

   /* mind a bemeneti, mind a kimeneti objektum adattípusa unsigned char lesz */
   kpds_set_attribute(in_object,  KPDS_VALUE_DATA_TYPE, KUBYTE);
   kpds_set_attribute(out_object, KPDS_VALUE_DATA_TYPE, KUBYTE);

   /* lekérdezzük a méret attribútumokat */
   kpds_get_attribute(in_object, KPDS_VALUE_SIZE, &w, &h, &d, &t, &e);
```

```
   /*
      létrehozunk egy-egy memóriaterületet a képi adatok számára, mindig le kell
      ellenőrizni, hogy sikerült-e létrehozni a memóratömböt, ha nem, akkor
      hibával ki kell lépni
   */
   plane = (unsigned char *)kmalloc(w*h*sizeof(unsigned char));
   res_plane = (char *)kmalloc(w*h*sizeof(unsigned char));
   if (!plane || !res_plane) {
     kerror(lib, rtn, "Could not allocate memory for the image\n");
     kexit(KEXIT_FAILURE);
  }
  /* -main_before_lib_call_end */

  /* -main_library_call */

  /*
    Most jön a képfeldolgozó eljárás érdemi része, bejárjuk a kép minden
    pixelét. Nem tudhatjuk, hogy álló vagy mozgó képről van szó, ezért a ct
    változóval bejáruk az összes frame-et, ha biztosak vagyunk benne, hogy a kép
    az kétdimenziós állókép, akkor ez elhagyható
  */
  for (ct = 0; ct < t; ct++) {
    kpds_set_attribute(in_object, KPDS_VALUE_POSITION, 0, 0, 0, ct, 0);
    kpds_get_data(in_object, KPDS_VALUE_PLANE, (kaddr)plane);
    kmemset(res_plane, 0, w*h*sizeof(char));

    for (ch = 0; ch < h; ch++) {
      for (cw = 0; cw < w; cw++) {
        pos = ch*w + cw;

      /* IRD IDE A KOD RESZLETET!!!! */
      /* csak a memória területekkel manipulálunk egyelőre */
      res_plane[pos] = 255    plane[pos]; /* itt számoljuk az inverzet */
      }
    }
    /* az eredményül kapott képi adatot betöltjük a kimeneti objektumba */
    kpds_set_attribute(out_object, KPDS_VALUE_POSITION, 0, 0, 0, ct, 0);
    kpds_put_data(out_object, KPDS_VALUE_PLANE, (kaddr)res_plane);
  }
  /* -main_library_call_end */

  /* -main_after_lib_call */
   /* beállítjuk a history string-et */
   if (!kpds_set_attribute(out_object, KPDS_HISTORY, kpds_history_string())) {
     kerror(lib,rtn,"Unable to set history on the destination object");
     kexit(KEXIT_FAILURE);
   }

   /* felszabadítjuk a lefoglalt memoriaterületet és lezárjuk az objektumokat */
   if (plane)
     kfree(plane);
   if (res_plane)
     kfree(res_plane);
   kpds_close_object(in_object);
   kpds_close_object(out_object);
  /* -main_after_lib_call_end */

   return TRUE;

}
```

Ha lefordítjuk, akkor következő eredményt kell kapnunk:



2. Kép transzponálása

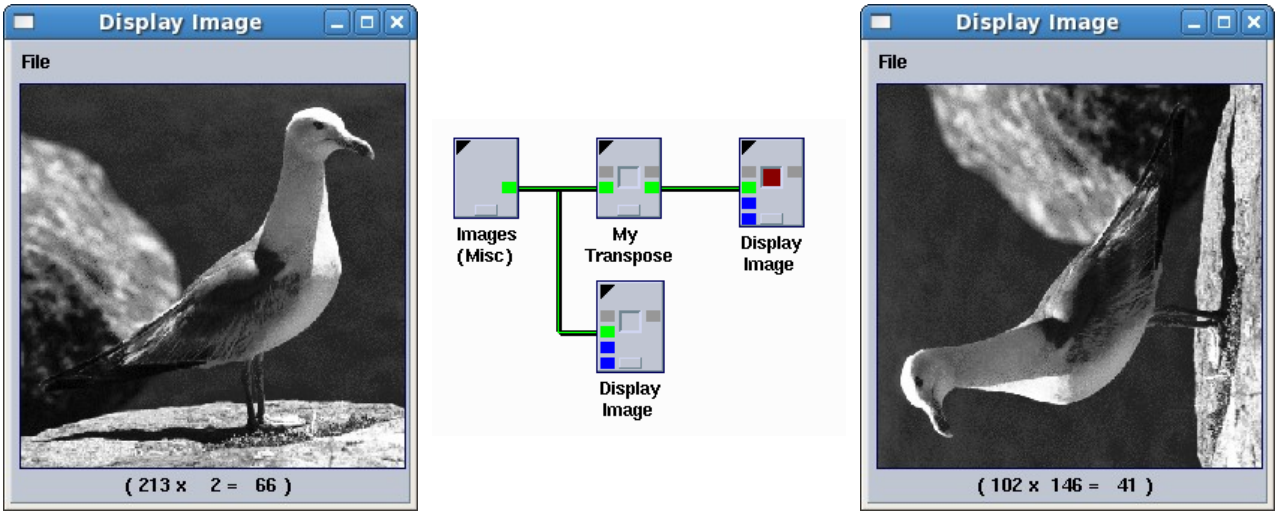
A kép transzponálása tulajdonképpen nem áll másból, minthogy felcseréljük a sor és oszlop koordinátákat. Figyelni kell arra, hogy az eredmény kép méretei is felcserélődnek, vagyis az eredmény kép szélessége akkora lesz, mint az eredeti magassága, és magassága pedig, mint az eredeti szélessége. A műveletet most is egy külön memóriaterületen hajtjuk végre, a tömb mérete így megegyezik, de az indexelésre figyelni kell!

```
int run_mytranspose(void)
{
   /*-- Put Your Code Here --*/


   /* -main_variable_list */
   char *lib = "mytranspose_obj";
   char *rtn = "main";
   kobject in_object = NULL;
   kobject out_object = NULL;
   unsigned char *plane;
   unsigned char *res_plane;
   int w, h, d, t, e;
   int cw, ch, ct, pos, res_pos;
/* -main_variable_list_end */

/* -main_before_lib_call */
   if ((in_object = kpds_open_input_object(clui_info->i_file))
          == KOBJECT_INVALID) {
     kerror(lib, rtn, "Can not open input object %s.\n", clui_info->i_file);
     kexit(KEXIT_FAILURE);
   }
   if ((out_object = kpds_open_output_object(clui_info->o_file))
        == KOBJECT_INVALID) {
     kerror(lib, rtn, "Can not open output object %s.\n", clui_info->o_file);
     kexit(KEXIT_FAILURE);
   }

   /*
     most nem másoljuk át az input objektumot az eredmény objektumba, ezért
     létre kell hozni a Value szegmenst az adat számára
   */
   kpds_create_value( out_object );
```

```
   kpds_set_attribute(in_object,  KPDS_VALUE_DATA_TYPE, KUBYTE);
   kpds_set_attribute(out_object, KPDS_VALUE_DATA_TYPE, KUBYTE);

  /* lekérdezzük az input kép méret attribútumát */
  kpds_get_attribute(in_object, KPDS_VALUE_SIZE, &w, &h, &d, &t, &e);


  /* ALLLITSD BE AZ OUTPUT OBJEKTUM MERETET!  */

  /*
     beállítjuk az output objektum méret attribútumát, a szélesség és magasság
     attribútumot felcseréljük
  */
  kpds_set_attribute( out_object, KPDS_VALUE_SIZE, h, w, d, t, e );

  /*
     lefoglaltjuk a memóriatömböket a képi adat számára,
     mindkét tömb ugyanakkora
  */
  plane = (unsigned char *)kmalloc(w*h*sizeof(unsigned char));
  res_plane = (char *)kmalloc(w*h*sizeof(unsigned char));
  if (!plane || !res_plane) {
     kerror(lib, rtn, "Could not allocate memory for the image\n");
     kexit(KEXIT_FAILURE);
  }


/* -main_before_lib_call_end */

/* -main_library_call */

   /* VALOSITSD MEG A TRANSZPONALAST!! */

   /*
     A transzponálás megvalósítása következik. Végigmegyünk az eredeti képen, és
     kiszámoljuk az indexeket. Az eredmény kép indexe különbözni fog a
     transzponáltétól
   */
   for ( ct = 0; ct <t; ct++ ) {
      kpds_set_attribute(in_object, KPDS_VALUE_POSITION, 0, 0, 0, ct, 0);
      kpds_get_data( in_object, KPDS_VALUE_PLANE, (kaddr)plane );
      kmemset(res_plane, 0, w*h*sizeof(char));

      for ( ch = 0; ch < h; ch++ ) {
        for ( cw = 0; cw < w; cw++ ) {
        pos = ch * w + cw;                /* az eredmény kép indexe */
        res_pos = cw * h + ch;            /* a transzponált kép indexe */
        res_plane[res_pos] = plane[pos];
      }
      }
      kpds_set_attribute(out_object, KPDS_VALUE_POSITION, 0, 0, 0, ct, 0);
      kpds_put_data(out_object, KPDS_VALUE_PLANE, (kaddr)res_plane);
   }

/* -main_library_call_end */

/* -main_after_lib_call */
   if (!kpds_set_attribute(out_object, KPDS_HISTORY, kpds_history_string())) {
     kerror(lib,rtn,"Unable to set history on the destination object");
     kexit(KEXIT_FAILURE);
   }
```

```
   if (plane)
     kfree(plane);
   if (res_plane)
     kfree(res_plane);
   kpds_close_object(in_object);
   kpds_close_object(out_object);
/* -main_after_lib_call_end */


   return TRUE;
}
```

Eredményül ezt kell kapnunk:



3. Kép simítása 3x3-as átlagoló szűrővel

A simításhoz kihasználunk egy technikai trükköt. A simításhoz gyakran számolni kell a környezeti indexeket. Ezeket a relatív indexeket le is tárolhatjuk egy tömbben.

-w-1	-w	-w+1
-1	p	1
w-1	w	w+1

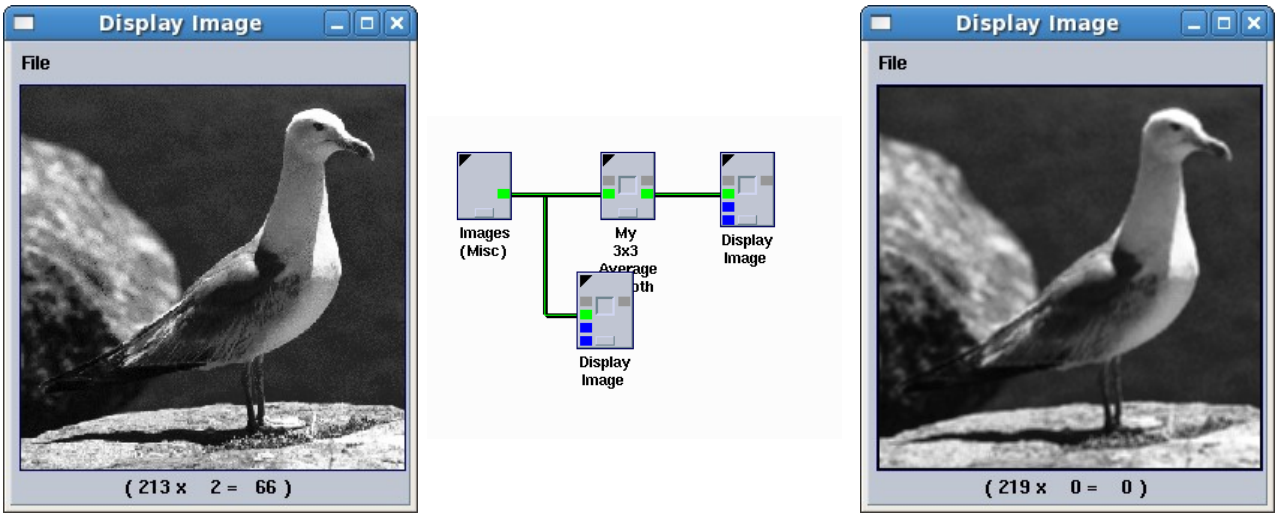
Ezt egy vektorba is emulálhatjuk, ha sorfolytonosan vesszük a környezeti indexeket. P helyére 0-t kell írni.
Ezt kihasználva nem kell mást tennünk, mint kiszámoljuk ebben a környezetben az átlagot, és azt írjuk be az eredmény képbe.

```
int run_my3x3averagesmooth(void)
{
   /*-- Put Your Code Here --*/

   /* -main_variable_list */
   char *lib = "mythreshold_obj";
   char *rtn = "main";
   kobject in_object = NULL;
   kobject out_object = NULL;
   unsigned char *plane;
   unsigned char *res_plane;
```

```
   int w, h, d, t, e;
   int cw, ch, ct, pos, i;
       int sum;

       int env3x3[9];

/* -main_variable_list_end */

/* -main_before_lib_call */
   if ((in_object = kpds_open_input_object(clui_info->i_file))
          == KOBJECT_INVALID) {
     kerror(lib, rtn, "Can not open input object %s.\n", clui_info->i_file);
     kexit(KEXIT_FAILURE);
   }
   if ((out_object = kpds_open_output_object(clui_info->o_file))
        == KOBJECT_INVALID) {
     kerror(lib, rtn, "Can not open output object %s.\n", clui_info->o_file);
     kexit(KEXIT_FAILURE);
   }
   if (!kpds_copy_object(in_object, out_object)) {
     kerror(lib, rtn, "Can not copy input object to output object.\n");
     kexit(KEXIT_FAILURE);
   }
   kpds_set_attribute(in_object,  KPDS_VALUE_DATA_TYPE, KUBYTE);
   kpds_set_attribute(out_object, KPDS_VALUE_DATA_TYPE, KUBYTE);

  kpds_get_attribute(in_object, KPDS_VALUE_SIZE, &w, &h, &d, &t, &e);

  plane = (unsigned char *)kmalloc(w*h*sizeof(unsigned char));
  res_plane = (char *)kmalloc(w*h*sizeof(unsigned char));
  if (!plane || !res_plane) {
     kerror(lib, rtn, "Could not allocate memory for the image\n");
     kexit(KEXIT_FAILURE);
  }
  /* beírjuk a tömbbe a környezeti indexeket */
      env3x3[0] = -w-1;
      env3x3[1] = -w;
      env3x3[2] = -w+1;
      env3x3[3] = -1;
      env3x3[4] = 0;
      env3x3[5] = 1;
      env3x3[6] = w-1;
      env3x3[7] = w;
      env3x3[8] = w+1;


/* -main_before_lib_call_end */

/* -main_library_call */
  for (ct = 0; ct < t; ct++) {
    kpds_set_attribute(in_object, KPDS_VALUE_POSITION, 0, 0, 0, ct, 0);
    kpds_get_data(in_object, KPDS_VALUE_PLANE, (kaddr)plane);
    kmemset(res_plane, 0, w*h*sizeof(char));

    for (ch = 1; ch < h-1; ch++) {
      for (cw = 1; cw < w-1; cw++) {
        pos = ch*w + cw;
            /* kiszámoljuk a környezet átlagát */
                       sum = 0;
                       for ( i = 0; i < 9; i++ ) {
                             sum += plane[pos + env3x3[i]];
                       }
                       sum /= 9;
                       res_plane[pos] = sum;
```

```
      }
    }

    kpds_set_attribute(out_object, KPDS_VALUE_POSITION, 0, 0, 0, ct, 0);
    kpds_put_data(out_object, KPDS_VALUE_PLANE, (kaddr)res_plane);
  }
/* -main_library_call_end */

/* -main_after_lib_call */
   if (!kpds_set_attribute(out_object, KPDS_HISTORY, kpds_history_string())) {
     kerror(lib,rtn,"Unable to set history on the destination object");
     kexit(KEXIT_FAILURE);
   }
   if (plane)
     kfree(plane);
   if (res_plane)
     kfree(res_plane);
   kpds_close_object(in_object);
   kpds_close_object(out_object);
/* -main_after_lib_call_end */

   return TRUE;
}
```

Eredményül ezt kell kapnunk:



4. RGB kép színkomponenseinek kinyerése

Az RGB kép esetében minden egyes pixel 3 komponensből áll, egy vörösből (R), egy zöldből (G), és egy kékből (B). Ezeket a komponenseket, mint vektort tudjuk kinyerni VisiQuest-ben.

```
int run_myrgbdecompose(void)
{
   /*-- Put Your Code Here --*/
       /* -main_variable_list */
   char *lib = "myrgbdecompose_obj";
   char *rtn = "main";
   kobject in_object = NULL;
   kobject red_object = NULL;
   kobject green_object = NULL;
   kobject blue_object = NULL;
```

```
   int inRGB[3];
   int red, green, blue;

   int redRGB[3];    /* segédtömb az eredmény komponenseknek */
   int greenRGB[3];  /* segédtömb az eredmény komponenseknek */
   int blueRGB[3];   /* segédtömb az eredmény komponenseknek */
   int w, h, d, t, e;
   int cw, ch, ct, pos;
/* -main_variable_list_end */

/* -main_before_lib_call */
   if ((in_object = kpds_open_input_object(clui_info->i_file))
          == KOBJECT_INVALID) {
     kerror(lib, rtn, "Can not open input object %s.\n", clui_info->i_file);
     kexit(KEXIT_FAILURE);
   }

   /* létrehozok három eredmény objektumot */
   if ((red_object = kpds_open_output_object(clui_info->ored_file))
        == KOBJECT_INVALID) {
     kerror(lib, rtn, "Can not open output object %s.\n", clui_info->ored_file);
     kexit(KEXIT_FAILURE);
   }
       if ((green_object = kpds_open_output_object(clui_info->ogreen_file))
        == KOBJECT_INVALID) {
     kerror(lib, rtn, "Can not open output object %s.\n", clui_info-
>ogreen_file);
     kexit(KEXIT_FAILURE);
   }
       if ((blue_object = kpds_open_output_object(clui_info->oblue_file))
        == KOBJECT_INVALID) {
     kerror(lib, rtn, "Can not open output object %s.\n", clui_info-
>oblue_file);
     kexit(KEXIT_FAILURE);
   }

   /* átmásolom az eredményekbe az inputot */
   if (!kpds_copy_object(in_object, red_object)) {
     kerror(lib, rtn, "Can not copy input object to output object.\n");
     kexit(KEXIT_FAILURE);
   }
   if (!kpds_copy_object(in_object, green_object)) {
     kerror(lib, rtn, "Can not copy input object to output object.\n");
     kexit(KEXIT_FAILURE);
   }
   if (!kpds_copy_object(in_object, blue_object)) {
     kerror(lib, rtn, "Can not copy input object to output object.\n");
     kexit(KEXIT_FAILURE);
   }

   /* mindegyik típusát unsigned int-re állítom */
   kpds_set_attribute(in_object,  KPDS_VALUE_DATA_TYPE, KUINT);
   kpds_set_attribute(red_object, KPDS_VALUE_DATA_TYPE, KUINT);
   kpds_set_attribute(green_object, KPDS_VALUE_DATA_TYPE, KUINT);
   kpds_set_attribute(blue_object, KPDS_VALUE_DATA_TYPE, KUINT);

   kpds_get_attribute(in_object, KPDS_VALUE_SIZE, &w, &h, &d, &t, &e);

/* -main_before_lib_call_end */

/* -main_library_call */

  for (ct = 0; ct < t; ct++) {
```

```
      for ( ch = 0; ch < h; ch++ ) {
        for ( cw = 0; cw < w; cw++  ) {
            kpds_set_attribute( in_object, KPDS_VALUE_POSITION, cw,ch,0, ct,0 );
            /* kinyerjük a komponens vektort az inputból */
            kpds_get_data( in_object, KPDS_VALUE_VECTOR, (kaddr)inRGB );

            pos = ch * w + cw;

            /*
               az eredmény vektorokat úgy állítjuk be, hogy a megfelelő
               komponens-vektorba csak az adott komponens kap 0-tól különböző
               értéket
            */
            redRGB[0] = inRGB[0];
            redRGB[1] = 0;
            redRGB[2] = 0;

            greenRGB[0] = 0;
            greenRGB[1] = inRGB[1];
            greenRGB[2] = 0;

            blueRGB[0] = 0;
            blueRGB[1] = 0;
            blueRGB[2] = inRGB[2];

            /* a komponens-vektorokat beírjuk az objektumokba */
            kpds_set_attribute( red_object, KPDS_VALUE_POSITION, cw,ch,0,ct,0);
            kpds_put_data( red_object, KPDS_VALUE_VECTOR, (kaddr)redRGB );
            kpds_set_attribute(green_object, KPDS_VALUE_POSITION, cw,ch,0,ct,0);
            kpds_put_data( green_object, KPDS_VALUE_VECTOR, (kaddr)greenRGB );
            kpds_set_attribute( blue_object, KPDS_VALUE_POSITION, cw,ch,0,ct,0);
            kpds_put_data( blue_object, KPDS_VALUE_VECTOR, (kaddr)blueRGB );

        }
      }
    }
/* -main_library_call_end */

/* -main_after_lib_call */

   if (!kpds_set_attribute(red_object, KPDS_HISTORY, kpds_history_string())) {
     kerror(lib,rtn,"Unable to set history on the destination object");
     kexit(KEXIT_FAILURE);
   }
   if (!kpds_set_attribute(green_object, KPDS_HISTORY, kpds_history_string())) {
     kerror(lib,rtn,"Unable to set history on the destination object");
     kexit(KEXIT_FAILURE);
   }
   if (!kpds_set_attribute(blue_object, KPDS_HISTORY, kpds_history_string())) {
     kerror(lib,rtn,"Unable to set history on the destination object");
     kexit(KEXIT_FAILURE);
   }

   kpds_close_object(in_object);
   kpds_close_object(red_object);
       kpds_close_object(green_object);
       kpds_close_object(blue_object);
/* -main_after_lib_call_end */
   return TRUE;
}
```

Eredményül a Mandril-os képre ezt kell kapnunk: