8. fejezet

ODE megoldás, tábla alapú szimulációk, ablakkezelés

Haladó Alkalmazott Programozás
ELTE, 2019

Közönséges differenciálegyenlet-rendszer (ODE) megoldás

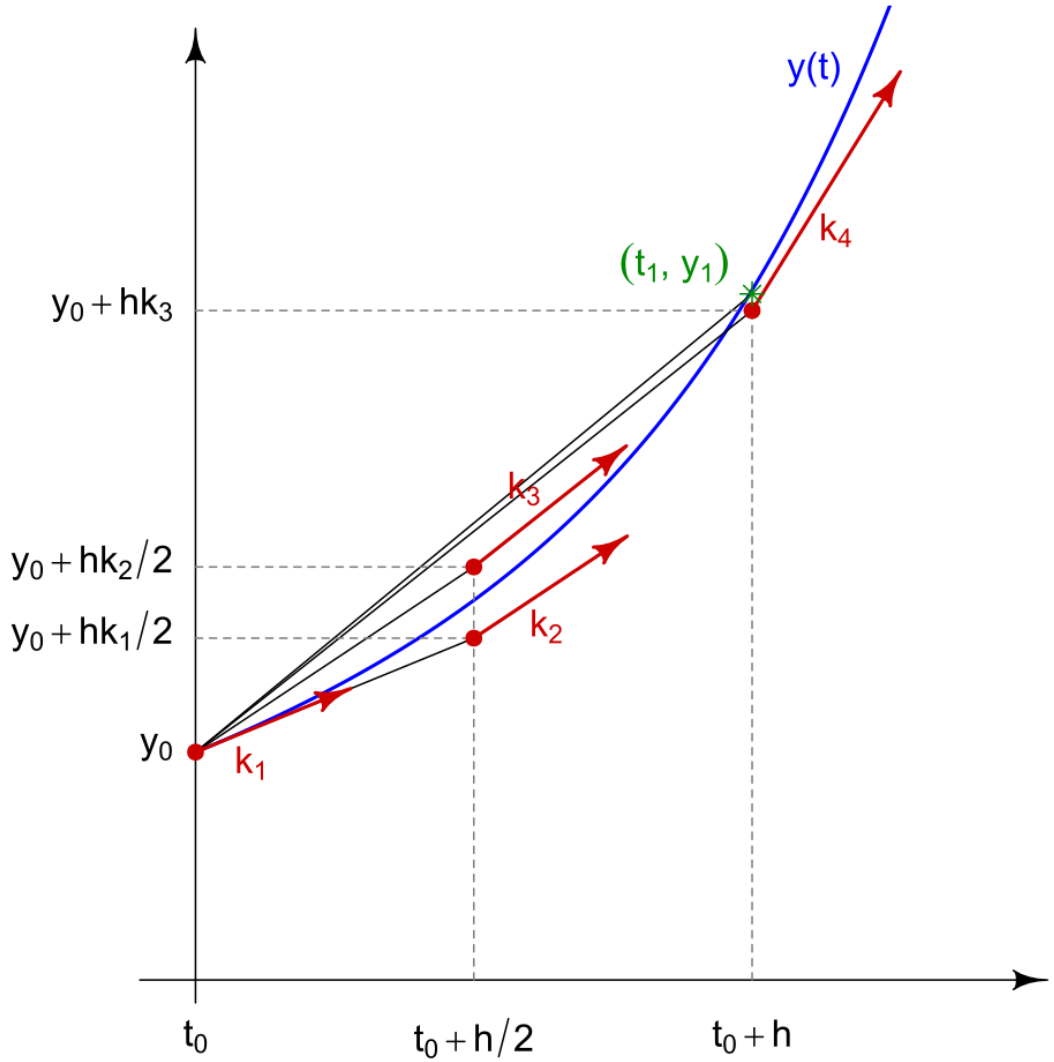
4-ed rendű Runge-Kutta

Lásd wiki.

Elsőrendű közönséges differenciálegyenlet-rendszert old meg, ismert kezdőfeltétellel.

Az egyenletrendszert az alábbi alakban tekinti:

$$\frac{d\vec{y}}{dt} = \vec{f}\,(t, \vec{y})$$



4-ed rendű Runge-Kutta

Lásd wiki. Egy lépés kódban így néz ki:

```
State k1 = f(t, y);
State k2 = f(t + h * (T)0.5, y + (h * (T)0.5) * k1);
State k3 = f(t + h * (T)0.5, y + (h * (T)0.5) * k2);
State k4 = f(t + h,          y +  h           * k3);

y = y + (k1 + k4 + (T)2 * (k2 + k3)) * (h / (T)6);
t = t + h;
```

4-ed rendű Runge-Kutta

A lépéseket addig ismételjük, amíg el nem érjük azt az időpontot, ameddig integrálni akarunk:

Kezdőfeltétel

```
T t = t0;
State y = y0;
while(t < t1)
{
  if(t + h > t1){ h = t1 - t; }
  State k1 = f(t, y);
  State k2 = f(t + h * (T)0.5, y + (h * (T)0.5) * k1);
  State k3 = f(t + h * (T)0.5, y + (h * (T)0.5) * k2);
  State k4 = f(t + h,          y +  h           * k3);
  y = y + (k1 + k4 + (T)2 * (k2 + k3)) * (h / (T)6);
  t = t + h;
}
```

4-ed rendű Runge-Kutta

A lépéseket addig ismételjük, amíg el nem érjük azt az időpontot, ameddig integrálni akarunk:

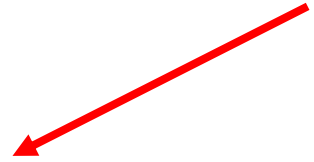
Megállási feltétel az időben

```
T t = t0;
State y = y0;
while(t < t1)
{
  if(t + h > t1){ h = t1 - t; }
  State k1 = f(t, y);
  State k2 = f(t + h * (T)0.5, y + (h * (T)0.5) * k1);
  State k3 = f(t + h * (T)0.5, y + (h * (T)0.5) * k2);
  State k4 = f(t + h,          y +  h           * k3);
  y = y + (k1 + k4 + (T)2 * (k2 + k3)) * (h / (T)6);
  t = t + h;
}
```

4-ed rendű Runge-Kutta

A lépéseket addig ismételjük, amíg el nem érjük azt az időpontot, ameddig integrálni akarunk:

```
T t = t0;
State y = y0;
while(t < t1)
{
  if(t + h > t1){ h = t1 - t; }
  State k1 = f(t, y);
  State k2 = f(t + h * (T)0.5, y + (h * (T)0.5) * k1);
  State k3 = f(t + h * (T)0.5, y + (h * (T)0.5) * k2);
  State k4 = f(t + h,          y +  h           * k3);
  y = y + (k1 + k4 + (T)2 * (k2 + k3)) * (h / (T)6);
  t = t + h;
}
```



Az utolsó lépésben ne lépjünk túl a határon!

4-ed rendű Runge-Kutta

A lépéseket addig ismételjük, amíg el nem érjük azt az időpontot, ameddig integrálni akarunk:

```
T t = t0;
State y = y0;
while(t < t1)
{
  if(t + h > t1){ h = t1 - t; }
  State k1 = f(t, y);
  State k2 = f(t + h * (T)0.5, y + (h * (T)0.5) * k1);
  State k3 = f(t + h * (T)0.5, y + (h * (T)0.5) * k2);
  State k4 = f(t + h,          y +  h           * k3);
  y = y + (k1 + k4 + (T)2 * (k2 + k3)) * (h / (T)6);
  t = t + h;
  cb(t, y);
}
```

Egy callback függvény, hogy a felhasználó értesülhessen a köztes lépések eredményeiről

4-ed rendű Runge-Kutta

```
template<typename State, typename T, typename RHS, typename Callback>
auto solve_rk4(State y0, T t0, T t1, T h, RHS f, Callback cb)
{
  T t = t0; State y = y0;
  while(t < t1)
  {
    if(t + h > t1){ h = t1 - t; }
    State k1 = f(t, y);
    State k2 = f(t + h * (T)0.5, y + (h * (T)0.5) * k1);
    State k3 = f(t + h * (T)0.5, y + (h * (T)0.5) * k2);
    State k4 = f(t + h,          y +  h           * k3);
    y = y + (k1 + k4 + (T)2 * (k2 + k3)) * (h / (T)6);
    t = t + h; cb(t, y);
  }
  return y;
}
```

4-ed rendű Runge-Kutta

Ebben most használhatjuk például a Vector2 típusunkat,
hogy egy két változós rendszer állapotát leírjuk.

```
Vector2<double> state, res;
double time = 0.0, h = 1e-4;
res = solve_rk4(state, time, time + 1.0, h,
      [](double t, Vector2<double> y)->Vector2<double>{ ... },
      [](double t, Vector2<double> y){ ... } );
```

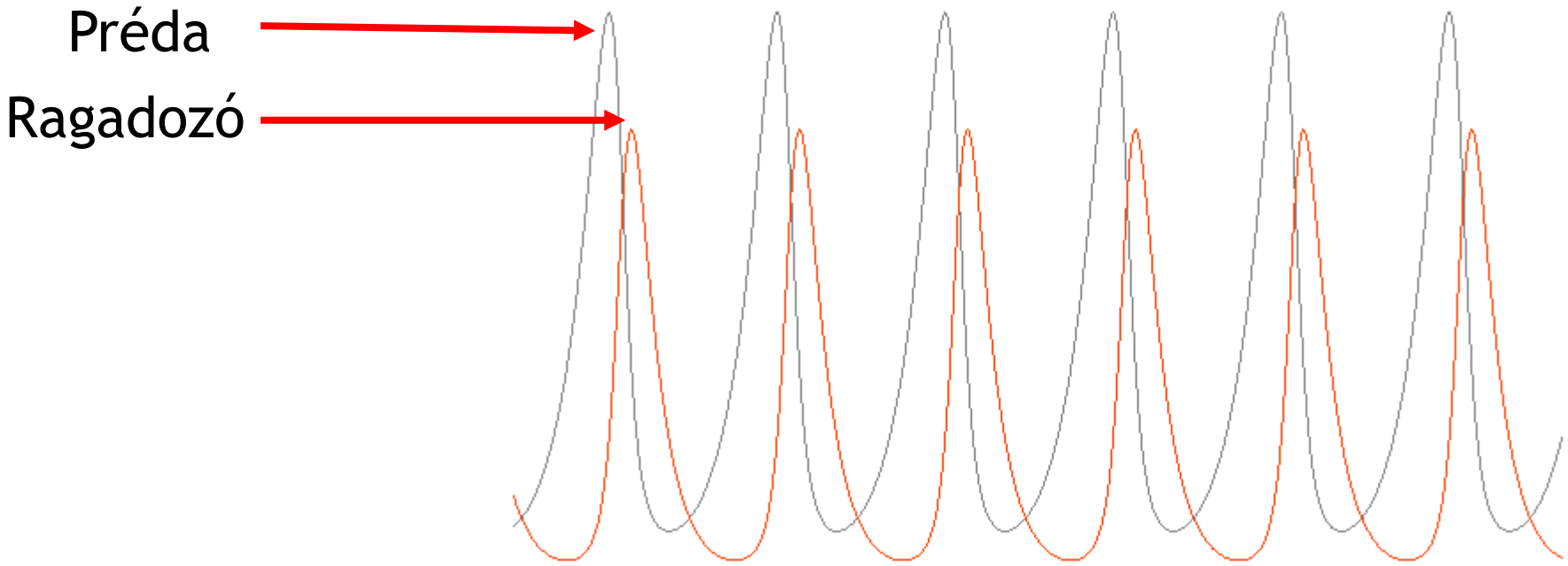
4-ed rendű Runge-Kutta

Példa:
Lotka-Volterra populáció dinamikai rendszer.

```
double a = 20., b = 1., c = 30.0, d = 1.;

[&](double t, Vector2<double> s)->Vector2<double>
{
  return {a*s.x      - b*s.x*s.y,
          d*s.x*s.y - c*s.y       };
},
```

4-ed rendű Runge-Kutta

Példa:
Lotka-Volterra populáció dinamikai rendszer.



Tábla alapú szimulációk

Tábla alapú szimulációk

Példa: Conway életjátéka (Wiki)

Adott egy 2D négyzetrács, ahol minden cellának két állapota van: élő, vagy halott. A celláknak a 8 legközelebbi szomszédjával van kölcsönhatása.

- Ha egy cella él, de kevesebb, mint 2, vagy több mint 3 szomszédja él, akkor elpusztul
- Ha egy cella halott, de pontosan 3 élő szomszédja van, akkor az élővé válik
- Minden más esetben változatlan az állapota.

Tábla alapú szimulációk

Ez egy tipikus példája a tábla alapú szimulációknak, ahol a következő képpen kell eljárnunk:

- Mivel egy cella új állapota nem csak az előző állapotától függ, ezért mindenképp két táblára van szükségünk! (dupla bufferelés)
- Ezek között váltakozik, hogy az egyikből olvasunk, másikba írunk, majd fordítva, ehhez egy indexet tartunk számon, ami 0-1 értéket vesz fel
- Az érdekes rész már csak az, hogy mit csinálunk a határokon...

Tábla alapú szimulációk


```
template<typename T>
struct Table2D
{
  std::vector<T> data;
  int w, h;

  T       & operator()(int x, int y)       { return data[y*w+x]; }
  T const& operator()(int x, int y) const { return data[y*w+x]; }

  template<typename F> void fill1(F&& f);
  template<typename F> void fill2(F&& f);
};
```

Tábla alapú szimulációk

Dupla buffer, 1 byte-ot foglal minden cella:

```
std::array<Table2D<char>, 2> table;
int idx = 0;
```

Random inicializálás:

```
std::mt19937 mt(42);
std::uniform_real_distribution<float> d(0.0, 1.0f);
table[0].fill1([&](int)->char{ return d(mt) < 0.5 ? 0 : 1; });
table[1].fill1([ ](int)->char{ return 0; });
```

Tábla alapú szimulációk

Léptetés:

Ebből olvasunk

```
table[1 - idx].fill2([&t = table[idx]](int x, int y)->char
  {
      auto c = t(x, y);
      ...
      return c;
  });
idx = 1 - idx;
```

Ebbe írunk

Megfordítjuk az indexeket, tehát a bufferek szerepét

Tábla alapú szimulációk

Léptetés:

Aktuális cella értéke

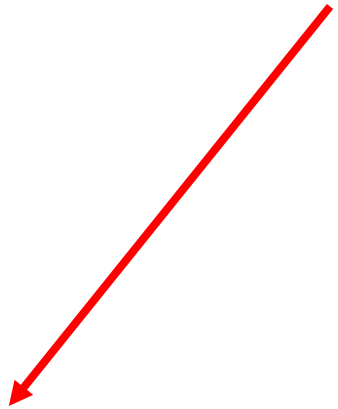
```
auto c = t(x, y);
auto xp1 = x == w-1 ? 0   : x+1;
auto xm1 = x == 0   ? w-1 : x-1;
auto yp1 = y == h-1 ? 0   : y+1;
auto ym1 = y == 0   ? h-1 : y-1;
auto sum = t(xm1, ym1) + t(x, ym1) + t(xp1, ym1)
         + t(xm1, y  ) +             t(xp1, y  )
         + t(xm1, yp1) + t(x, yp1) + t(xp1, yp1);
...
```

x-y szomszédok koordinátái, periodikus határfeltétellel!

Tábla alapú szimulációk

Léptetés:

A 8 szomszéd állapotának összege
(mindegyik hívás 1-et ad ha él, 0-t ha halott)



```
auto c = t(x, y);
auto xp1 = x == w-1 ? 0   : x+1;
auto xm1 = x == 0   ? w-1 : x-1;
auto yp1 = y == h-1 ? 0   : y+1;
auto ym1 = y == 0   ? h-1 : y-1;
auto sum = t(xm1, ym1) + t(x, ym1) + t(xp1, ym1)
         + t(xm1, y  ) +             t(xp1, y  )
         + t(xm1, yp1) + t(x, yp1) + t(xp1, yp1);
...
```

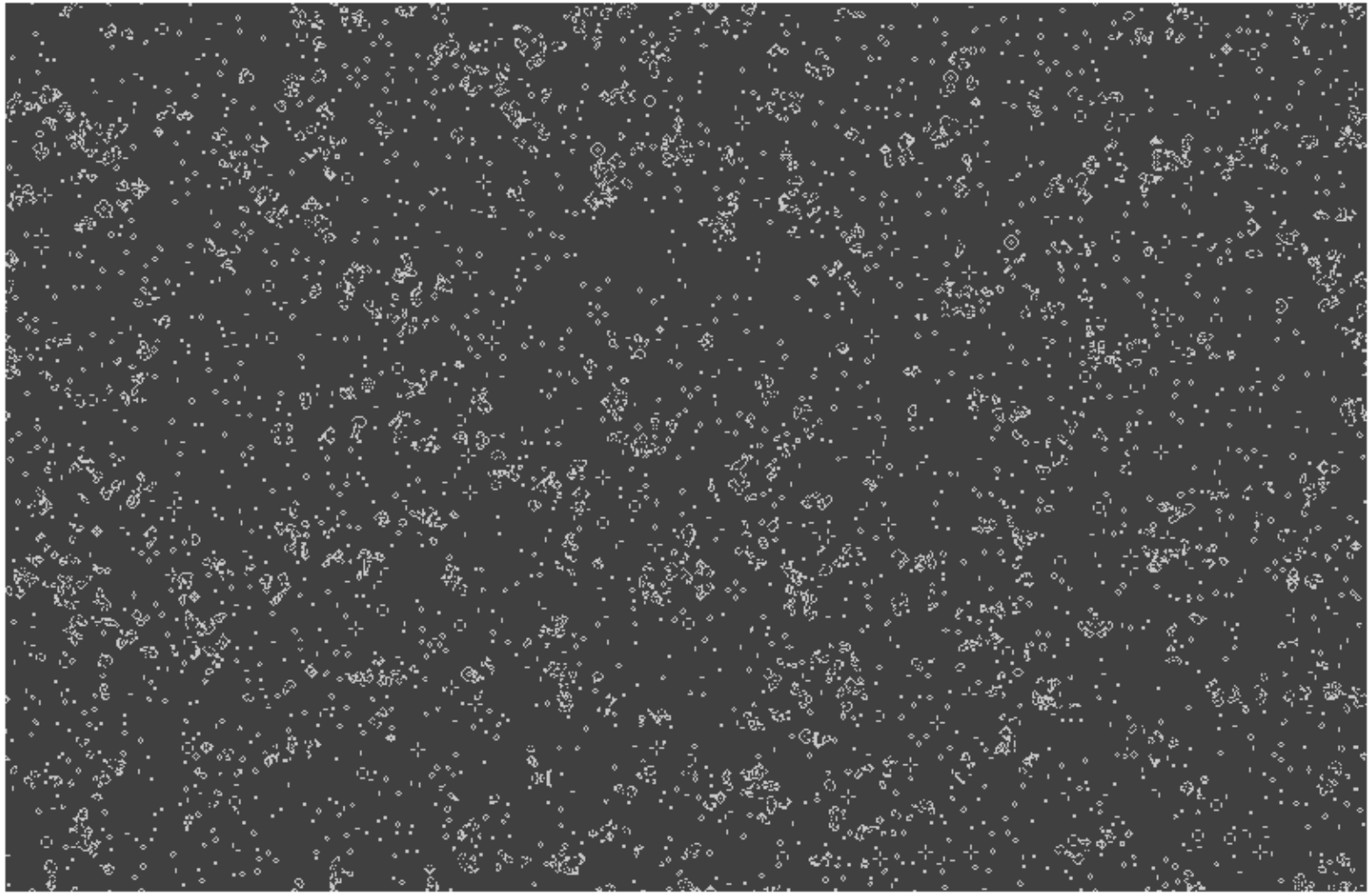
Tábla alapú szimulációk

Léptetés:

```
...
auto sum = t(xm1, ym1) + t(x, ym1) + t(xp1, ym1)
         + t(xm1, y  ) +             t(xp1, y  )
         + t(xm1, yp1) + t(x, yp1) + t(xp1, yp1);

if(c == 0 &&  sum == 3             ){ return 1; }
if(c == 1 && (sum < 2 || sum > 3)){ return 0; }
return c;
```

Állapot átmenetek a szabályok szerint

Tábla alapú szimulációk



- Ablakkezelés

Ablakkezelés

A C++ Standard Library-nak nem része semmilyen grafikai, vagy ablak kezelési szolgáltatás. Ezért ezeket külső könyvtárakon keresztül tudjuk csak megvalósítani.

Néhány elterjedt könyvtár:

- Qt
- SFML
- SDL

Ablakkezelés

Problémák:

- A dependenciák kezelése C++-ban nem megoldott, ezért a függőségeket sokszor nehézkes beállítani, telepíteni (különösen Windows alatt)
- Ezek a könyvtárak régi tervezésűek és bár fokozatosan modernizálják őket, nem tudnak lépést tartani az új C++ szolgáltatásokkal, tervezési koncepciókkal
- Egyszerű feladatokra feleslegesen nagyok és bonyolultak

Ablakkezelés

A mostani óra kedvéért nézzünk egy egyszerű ablakozó rendszert és egy szoftveres rajzolót, amivel minimális munkával tudunk egyszerű ábrákat megjeleníteni C++-ból egy ablakban.

miniwnd

Ablakkezelés

CMakeLists.txt: új elemek:

```
if (UNIX)
  find_package(X11 REQUIRED)
endif ()

target_include_directories (${PROJECT_NAME} PRIVATE
                            $<$<PLATFORM_ID:Linux>:${X11_INCLUDE_DIR}>)

if (UNIX)
  target_link_libraries (${PROJECT_NAME} PRIVATE ${X11_LIBRARIES})
endif ()

target_compile_definitions(${PROJECT_NAME} PRIVATE
                           $<$<CXX_COMPILER_ID:MSVC>:_UNICODE UNICODE>)
```

Linux / UNIX alatt az X11 könyvtárat használjuk, ennek a telepítési helyét meg tudja keresni a CMake

Ablakkezelés

CMakeLists.txt: új elemek:

Linux / UNIX alatt az X11 könyvtárat headerjeit hozzá kell adni a fordításhoz

```
if (UNIX)
  find_package(X11 REQUIRED)
endif ()

target_include_directories (${PROJECT_NAME} PRIVATE
                            $<$<PLATFORM_ID:Linux>:${X11_INCLUDE_DIR}>)

if (UNIX)
  target_link_libraries (${PROJECT_NAME} PRIVATE ${X11_LIBRARIES})
endif ()

target_compile_definitions(${PROJECT_NAME} PRIVATE
                           $<$<CXX_COMPILER_ID:MSVC>:_UNICODE UNICODE>)
```

Ablakkezelés

CMakeLists.txt: új elemek:

Linux / UNIX alatt az X11 könyvtárat binárisát hozzá kell linkelni a targethez

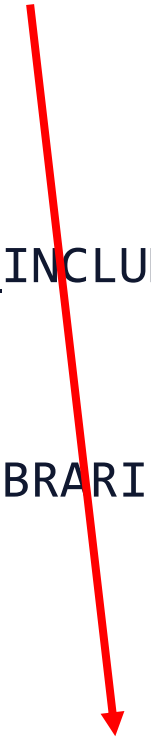
```
if (UNIX)
  find_package(X11 REQUIRED)
endif ()

target_include_directories (${PROJECT_NAME} PRIVATE
                            $<$<PLATFORM_ID:Linux>:${X11_INCLUDE_DIR}>)


if (UNIX)
  target_link_libraries (${PROJECT_NAME} PRIVATE ${X11_LIBRARIES})
endif ()

target_compile_definitions(${PROJECT_NAME} PRIVATE
                           $<$<CXX_COMPILER_ID:MSVC>:_UNICODE UNICODE>)
```

Ablakkezelés

CMakeLists.txt: új elemek:

```
if (UNIX)
  find_package(X11 REQUIRED)
endif ()

target_include_directories (${PROJECT_NAME} PRIVATE
                            $<$<PLATFORM_ID:Linux>:${X11_INCLUDE_DIR}>)

if (UNIX)
  target_link_libraries (${PROJECT_NAME} PRIVATE ${X11_LIBRARIES})
endif ()

target_compile_definitions(${PROJECT_NAME} PRIVATE
                           $<$<CXX_COMPILER_ID:MSVC>:_UNICODE UNICODE>)
```

Windows alatt érdemes az UNICODE kódolást és az ahhoz tartozó API-t használni



Ablakkezelés

Minimális ablak osztály:

```
#include "miniwindow.h"

int main()
{
  MainWindow wnd;

  bool res = wnd.open(L"Window name", {64, 64}, {640, 480},
             true, [&]{ return true; });

  return res ? 0 : -1;
}
```

Ablakkezelés

Minimális ablak osztály:

```
#include "miniwindow.h"

int main()
{
  MainWindow wnd;

  bool res = wnd.open(L"Window name", {64, 64}, {640, 480},
             true, [&]{ return true; });

  return res ? 0 : -1;
}
```

Az ablak objektum

Az ablak létrehozása

Ablakkezelés

Minimális ablak osztály:

```
#include "miniwindow.h"

int main()
{
  MainWindow wnd;

  bool res = wnd.open(L"Window name", {64, 64}, {640, 480},
             true, [&]{ return true; });

  return res ? 0 : -1;
}
```

Az ablak felirata, kezdőpozíciója, kezdőmérete

Ablakkezelés

Minimális ablak osztály:

```
#include "miniwindow.h"

int main()
{
  MainWindow wnd;

  bool res = wnd.open(L"Window name", {64, 64}, {640, 480},
             true, [&]{ return true; });

  return res ? 0 : -1;
}
```

Legyen-e dekoráció (címsor, gombok)

Ablakkezelés

Minimális ablak osztály:

```
#include "miniwindow.h"

int main()
{
  MainWindow wnd;

  bool res = wnd.open(L"Window name", {64, 64}, {640, 480},
             true, [&]{ return true; });

  return res ? 0 : -1;
}
```

Segéd inicializációs függvény, ami lefut, mielőtt az ablak létrehozása befejeződik (most nem kell nekünk)

Ablakkezelés

Minimális ablak osztály:

```
#include "miniwindow.h"

int main()
{
  MainWindow wnd;

  bool res = wnd.open(L"Window name", {64, 64}, {640, 480},
             true, [&]{ return true; });

  return res ? 0 : -1;
}
```

Az ablak az open függvényben végtelen ciklusba fut: ebben kezeli le a külső eseményeket és végzi a rajzolást. Ez a függvény csak akkor tér vissza, ha becsuktuk az ablakot, vagy valami hiba történt!

Ablakkezelés

Esemény kezelés:

```
MainWindow wnd;

wnd.mouseHandler([&](Mouse const& m){ });
wnd.resizeHandler([&](int w, int h, StateChange sc){ } );
wnd.idleHandler([&]{ });
wnd.exitHandler([&]{ });
wnd.renderHandler( [&](SoftwareRenderer& r){ });

bool res = wnd.open(...);
```

Ablakkezelés

Esemény kezelés:

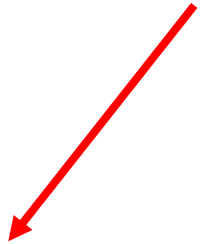
A külső eseményekről vissza hívó függvényeken keresztül értesülhetünk, de ezeket meg kell adnunk, mielőtt az open-el megnyitjuk az ablakot!

```
MainWindow wnd;

wnd.mouseHandler([&](Mouse const& m){ });
wnd.resizeHandler([&](int w, int h, StateChange sc){ } );
wnd.idleHandler([&]{ });
wnd.exitHandler([&]{ });
wnd.renderHandler( [&](SoftwareRenderer& r){ });

bool res = wnd.open(...);
```

Ablakkezelés

Esemény kezelés:

Ez a függvény hívódik meg,
ha az egérrel történik valami



```
MainWindow wnd;

wnd.mouseHandler([&](Mouse const& m){ });
wnd.resizeHandler([&](int w, int h, StateChange sc){ } );
wnd.idleHandler([&]{ });
wnd.exitHandler([&]{ });
wnd.renderHandler( [&](SoftwareRenderer& r){ });

bool res = wnd.open(...);
```

Ablakkezelés

Esemény kezelés:

Ez a függvény hívódik meg,
ha az ablakkal történik valami
(átméretezik, vagy minimalizálják,
újra felnyitják stb.)

```
MainWindow wnd;

wnd.mouseHandler([&](Mouse const& m){ });
wnd.resizeHandler([&](int w, int h, StateChange sc){ } );
wnd.idleHandler([&]{ });
wnd.exitHandler([&]{ });
wnd.renderHandler( [&](SoftwareRenderer& r){ });

bool res = wnd.open(...);
```

Ablakkezelés

Esemény kezelés:

```
MainWindow wnd;

wnd.mouseHandler([&](Mouse const& m){ });
wnd.resizeHandler([&](int w, int h, StateChange sc){ } );
wnd.idleHandler([&]{ });
wnd.exitHandler([&]{ });
wnd.renderHandler( [&](SoftwareRenderer& r){ });

bool res = wnd.open(...);
```

Ez a függvény hívódik meg,
amikor az ablaknak éppen semmi más
dolga nincs, minden eseményt lekezelt
Itt érdemes léptetni a szimulációt.

Ablakkezelés

Esemény kezelés:

Ez a függvény hívódik meg utoljára,
mielőtt az ablak visszatér az open függvényből

```
MainWindow wnd;

wnd.mouseHandler([&](Mouse const& m){ });
wnd.resizeHandler([&](int w, int h, StateChange sc){ } );
wnd.idleHandler([&]{ });
wnd.exitHandler([&]{ });
wnd.renderHandler( [&](SoftwareRenderer& r){ });

bool res = wnd.open(...);
```

Ablakkezelés

Esemény kezelés:

Ez a függvény hívódik meg,
amikor az ablakot újra kell rajzolni

```
MainWindow wnd;

wnd.mouseHandler([&](Mouse const& m){ });
wnd.resizeHandler([&](int w, int h, StateChange sc){ } );
wnd.idleHandler([&]{ });
wnd.exitHandler([&]{ });
wnd.renderHandler( [&](SoftwareRenderer& r){ });

bool res = wnd.open(...);
```

Ablakkezelés

Esemény kezelés:

Konkrét példák a példakódokban.

Ablakkezelés

A szoftveres renderelő szolgáltatásai jelenleg:

```
struct SoftwareRenderer
{
  Image2D backbuffer;
  void resize(int w, int h);
  void setpixel(int x, int y, Color c);
  Color getpixel(int x, int y);
  template<typename F> void forall_pixels(F&& f);
  template<typename T, typename F>
  void lineplot(int x, int y, int w, int h,
                T xmin, T xmax, T ymin, T ymax, Color col, F&& f);
  template<typename F>
  void line(int x0, int y0, int x1, int y1, F&& f);
  template<typename T, typename F>
  void triangle(T x0, T y0, T x1, T y1, T x2, T y2, F&& f);
  template<typename F>
  void ellipse(int xm, int ym, int a, int b, F&& f)

};
```

Ablakkezelés


```
struct SoftwareRenderer
{
  Image2D backbuffer;
  void resize(int w, int h);
  void setpixel(int x, int y, Color c);
  Color getpixel(int x, int y);
  template<typename F> void forall_pixels(F&& f);
  template<typename T, typename F>
  void lineplot(int x, int y, int w, int h,
                T xmin, T xmax, T ymin, T ymax, Color col, F&& f);
  template<typename F>
  void line(int x0, int y0, int x1, int y1, F&& f);
  template<typename T, typename F>
  void triangle(T x0, T y0, T x1, T y1, T x2, T y2, F&& f);
  template<typename F>
  void ellipse(int xm, int ym, int a, int b, F&& f)
};
```

Átméretezés

Pixelek beállítása, kiolvasása

Minden pixel módosítása koordináta alapján

Vonal ábra egy tetszőleges f(x) függvényből

Vonal két pont között

Háromszög 3 ponttal megadva

Ellipszis (középpont és a vízszintes, függőleges átmérő)

Ablakkezelés

```
struct SoftwareRenderer
{
  template<typename F>
  void line(int x0, int y0, int x1, int y1, F&& f);
  template<typename T, typename F>
  void triangle(T x0, T y0, T x1, T y1, T x2, T y2, F&& f);
  template<typename F>
  void ellipse(int xm, int ym, int a, int b, F&& f)
};
```

Ezeknél f az előző színt kapja meg, és az új színt adja vissza!

Egyszerű könyvtárak kép formátumok kezeléséhez, rajzoláshoz, és sok egyébhez:

stb

Mégtöbb: itt

Beadandó feladatok

link