SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 211

POSTUPCI MODELIRANJA PROČELJA ZGRADA TEMELJENIH NA FOTOGRAFIJAMA

Mirjana Ostojić

Zagreb, lipanj 2011.

Sadržaj

Uvod

Modeliranje pročelja urbanih okruženja posljednjih godina postaje sve popularnije. Porastom popularnosti računalnih igara, filmova i popularnih servisa poput *Google Earth-a* i *Microsoft Visual Earth-a*, u kojima se nastoji prikazati što vjerniji prikaz urbanih okruženja, raste i broj tehnika koje pokušavaju što vjernije i efikasnije prikazati virtualni grad. Kako bi se postigao fotorealističan izgled stvorenog modela koriste se uglavnom obrađene fotografije okruženja koja se modeliraju.

U nastavku će biti predstavljen rad na aplikaciji kojom se nastoji što bolje prikazati urbana okruženja. Prije opisa rada aplikacije i rezultata koji su dobiveni, objasnit će se teorija na kojoj je rad temeljen, a uključuje rad na računalnoj grafici, animaciji i računalnom vidu, te digitalnoj obradi slike.

1. Postupci modeliranja urbanih okruženja

Najopćenitija podjela postupaka modeliranja urbanih okruženja jest podjela na:

- postupke koji koriste fotografije snimljene iz zraka,
- postupke koji koriste fotografije snimljene na ulicama grada i
- postupke koji koriste obje vrste fotografija.

Ovisno o razini detalja koja se želi postići, koriste se fotografije manje ili veće razlučivosti.

Postupci modeliranja urbanih okruženja korištenjem fotografija snimljenih iz zraka daju zadovoljavajuće rezultate ako nije bitno koliko su detaljni prikazi ulica grada. Npr. ako koristimo *Google Earth* i želimo pronaći svoju kuću, moći ćemo ju pronaći, ali ćemo ju promatrati isključivo iz ptičje perspektive i što se više približavamo kući, njen prikaz je sve mutniji.

U računalnim igrama, gdje se radnja odvija u poznatim gradovima i likovi se kreću ulicama, potrebno je osmisliti postupak koji će dovoljno detaljno prikazivati zgrade ulica kako bi se stvorio realistični prikaz. Nedostatak takvih postupaka je najčešće potreba za obradom velikog broja fotografija velikih dimenzija i razlučivosti koje bi davale dovoljno detalja, a osim što ih je teško obrađivati, nije jednostavno niti prikupiti fotografije cijelog grada.

S obzirom na dosadašnje objavljene radove s konferencija i simpozija za računalnu grafiku i računalni vid, postupke modeliranja urbanih okruženja možemo podijeliti na četiri metode:

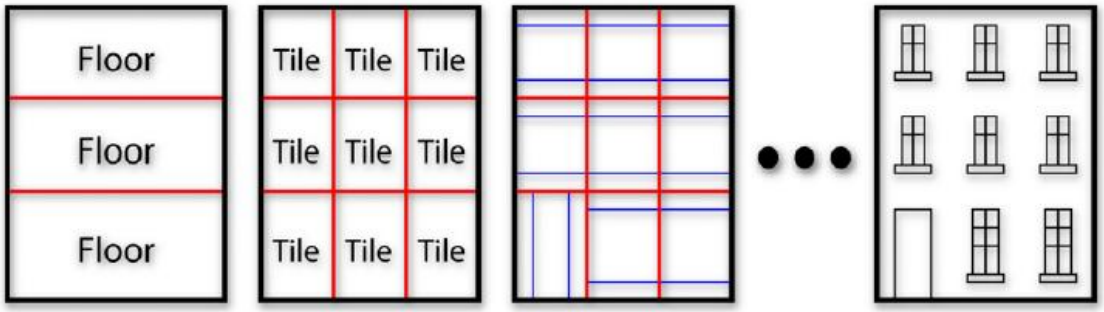
1. metode temeljene na pravilima
2. metode temeljene na fotografijama
3. metode temeljene na računalnom vidu
4. kombinacija navedenih metoda

1.1. Metode temeljene na pravilima

Metode temeljene na pravilima uglavnom se odnose na metode inspirirane gramatikom i L-sustavima. Ovisno o rezultatu koji želimo postići te razini detalja zgrada, odrede se specifična pravila koja se primjenjuju na podacima.

Nedostatak ovakve metode jest težina definiranja specifičnih pravila kako bi se generirale određene građevine, npr. za građevine pravokutnog oblika sa sličnim ili ponavljajućim katovima, veličinama prozora, balkona i vrata, vrijedit će vjerojatno jedan skup pravila, dok će se za cilindarske oblike građevina, poput Ciboninog tornja, morati koristiti drugačiji skup pravila.

Najčešće se primjenom određenih pravila radi podjela građevine na katove, odjeljke (*engl. tile*) na katovima, te prozore unutar odjeljaka [9]. Najpoznatiji radovi koji su koristili metode temeljene na pravilima spomenuti su u: [2] i [9].



Slika 1. Prikaz sustava koji računa hijerarhijsku strukturu pročelja.

1.2. Metode temeljene na fotografijama

Metode temeljene na fotografijama koriste se fotografijama kako bi generirale modele građevina. Najčešće je kod takvih metoda potrebna i interakcija korisnika, iako se većinu opcija nastoji automatizirati. Takva interaktivnost je često zamorna za korisnika, pogotovo ako se koristi veća količina ulaznih fotografija, a za većinu treba tražiti određenu informaciju od korisnika.

Slično kao i kod metoda temeljenih na pravilima, i neke metode temeljene na fotografijama neće imati isti učinak kod različitih građevina. Primjerice, kod zgrada koje imaju jednostavna pročelja moguće je automatski odrediti dijelove pročelja a od korisnika zatražiti manualno upisivanje dubine dijelova čime će se ostvariti zadovoljavajući model zgrade. Ipak, za kompliciranije građevine potrebna je veća interakcija korisnika pa je tehnika i manje privlačna korisniku.

Najpoznatiji radovi koji su se koristili metodama temeljenim na fotografijama su: [1], [4] i [8].

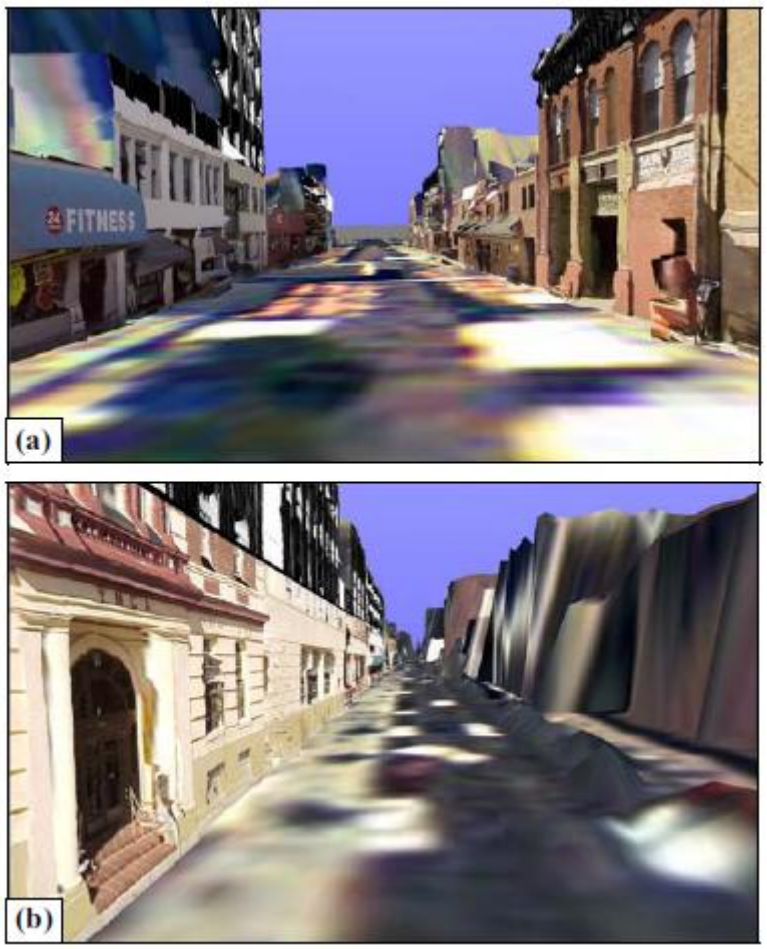
Slika 2. Rezultat metode temeljene na ulaznim fotografijama [1]. Pri dnu su ulazne fotografije koje sustav prima, u sredini je generirani model niza slika, a na samom vrhu približeni detalji generiranog modela.

1.3. Metode temeljene na računalnom vidu

Metode temeljene na računalnom vidu automatski konstruiraju modele građevina iz fotografija. Korištenjem algoritama tipičnih za računalni vid, poput detekcija obrisa, rubova te različitih oblika na fotografijama, prepoznavanje dijelova fotografije koji se repetitivno pojavljuju i ostalih algoritama, dobivaju se podaci kojima se bez pomoći korisnika mogu rekonstruirati scene viđene na ulaznim fotografijama. Također, računalnim vidom moguće je iz video podataka izlučiti potrebne podatke i kreirati vjeran model.

Kao i u prethodnim tehnikama, što imamo bolje podatke, poput fotografija i video prikaza visoke razlučivosti, dobit ćemo i bolji rezultat. S obzirom da je zanimanje za računalni vid u porastu i s obzirom da ovakve metode daju rezultate koji su kvalitetni ali i više automatizirani, metode temeljene na računalnom vidu se razvijaju puno brže nego ostale.

Poznati radovi temeljeni na ovakvim metodama su: [3], [12] i [10].



Slika 3. Prikaz modela ulice i pročelja zgrada nastao metodom temeljenom na računalnom vidu

1.4. Kombinacija navedenih metoda

Kombinacijom više metoda najčešće se ostvaruju bolji rezultati nego što bi bili korištenjem samo jedne metode. Ovisno o zadanom cilju koriste se metode koje su najpogodnije za obavljanje pojedine zadaće. Za automatsko određivanje strukture građevina, poput prozora i vrata, najčešće se koriste metode računalnog vida. Za efikasnu i optimalnu izgradnju građevine koriste se metode temeljene na pravilima, a za što realističniji dojam scene zaslužne su metode temeljene na fotografijama. Iako je većina radova spomenutih u prethodnim potpoglavljima većim dijelom orijentirana na neku od metoda, skoro svaki rad sadrži i neke karakteristike ostalih metoda, pa se može reći i da su njihovi rezultati temeljeni kombinacijom više metoda.

2. Postupak modeliranja urbanih okruženja temeljen na fotografijama – implementacija

U sklopu ovog rada izrađena je implementacija metode modeliranja urbanih okruženja temeljena na fotografijama. Implementacija je zamišljena kao programska aplikacija kojoj korisnik kao ulazne podatke predaje fotografije pročelja, a zatim se, uz određenu korisničku interakciju, kao rezultat dobiva vizualni prikaz 3D modela zgrade s pročeljem poznatim s ulaznih fotografija.

Implementacija se sastoji od nekoliko koraka:

1. Predaja ulaznih fotografija
2. Obrada ulaznih fotografija
3. Prikaz i modifikacija obrade ulaznih podataka
4. Dekompozicija pročelja
5. Kreiranje dubinske mape
6. Izrada 3D modela.

Predaja ulaznih fotografija. Korisnik predaje sustavu fotografije pročelja kao ulazne podatke. Poželjno je da su ulazne fotografije snimljene s kamerom postavljene nasuprot odabranog pročelja, jer je na taj način najlakše izlučiti potrebne podatke, a i dobiva se bolji rezultat. Prije predaje ulaznih fotografija potrebno je izrezati samo one dijelove fotografija koji predstavljaju pročelja jer su jedino takvi podaci korisni sustavu. Npr. poželjno je iz originalnih fotografija izrezati prikaz dijelova neba ili ceste.

Obrada ulaznih fotografija. Sustav obrađuje ulazne fotografije koristeći različite metode obrade fotografija i dobiva niz korisnih podataka koji su neophodni za izvršavanje ostalih koraka.

Prikaz i modifikacija obrade ulaznih podataka. Sustav prikazuje podatke dobivene obradom ulaznih fotografija i omogućava korisniku da samostalno ispravi obradu fotografija ako automatska obrada nije zadovoljavajuća ili potpuna.

Dekompozicija pročelja. Sustav automatski odrađuje dekompoziciju pročelja dobivenog s fotografija, odnosno, izdvoji elemente pročelja poput vrata, balkona i prozora.

Kreiranje dubinske mape. Sustav kreira dubinsku mapu i omogućava korisniku da samostalno ispravi dubine pojedinih elemenata ako automatska izrada dubina nije zadovoljavajuća ili potpuna.

Izrada 3D modela. Sustav na temelju dobivenih informacija izrađuje 3D modele zgrada sa pripadajućim teksturama te omogućava korisniku kretanje po kreiranoj sceni.

Slika 4. Redoslijed koraka modeliranja zgrada temeljenih na fotografijama

3. Obrada ulaznih fotografija

3.1. Detekcija rubova

Detekcija rubova na fotografijama jedan je od osnovnih problema u digitalnoj obradi slike, ali često i najkorisnija operacija na slici.

Rubovi su područja na slici s velikim razlikama u intenzitetu točaka i predstavljaju granice objekata što se može iskoristiti za prepoznavanje objekata, detekciju položaja objekta u slici i detekciju orijentacije objekta.

Detekcija rubova smanjuje količinu podataka i filtrira sve manje potrebne informacije ostavljajući samo rubove.

Postoji mnogo metoda detekcije rubova, no većina se može podijeliti na **gradijentne** i **Laplaceove metode**. Gradijentne metode traže maksimume i minimume u prvoj derivaciji slike, dok Laplaceove metode traže nule u drugoj derivaciji slike.

3.1.1. Cannyjev algoritam detekcije rubova

Jedan od najučinkovitijih, ali i najjednostavnijih algoritama koji uspješno rješavaju problem detekcije rubova jest Cannyjev algoritam detekcije rubova. Cannyjev algoritam pripada skupini gradijentnih metoda.

Algoritam je osmislio John F. Canny 1986. godine i predstavio u radu "*A Computational Approach to Edge Detection*". Prema Cannyju, dobar detektor rubova treba imati sljedeće karakteristike:

- *dobra detekcija* – treba detektirati što više rubova na slici, odnosno, maksimizirati odnos signala i šuma.
- *dobra lokalizacija položaja ruba* – udaljenost stvarnog ruba od detektiranog ruba treba biti što manja.

- *minimalan broj odziva na jedan rub* – postojeći rub u slici trebao bi biti označen samo jednom, odnosno, šumovi ne bi smjeli kreirati lažne rubove.

Cannyjev algoritam detekcije rubova sastoji se od nekoliko koraka:

1. Smanjenje šumova
2. Izračun magnitude i kuta gradijenta
3. Potiskivanje slikovnih elemenata koji nemaju maksimalan gradijent (engl. *Non-Maximum Surpression)*
4. Histereza pragova (engl. *Hysteresis Thresholding*)

3.1.1.1. Smanjenje šumova

Smanjenjem šumova na slici (engl. *Noise reduction*) nastoji se olakšati 'posao' detektoru rubova kako bi se ostvarila karakteristika minimalnog odziva. Pomoću Gaussove maske obavlja se konvolucija slike kako bi se uklonio šum.

$$\mathbf{B} = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * \mathbf{A}.$$

Slika 5. Primjer matrice koja se koristi kao Gaussov filter.

3.1.1.2. Izračun magnitude i kuta gradijenta

Područja maksimuma prve derivacije određuju se korištenjem operatora. Uglavnom se koriste četiri operatora: dva operatora koji određuju gradijente u horizontalnom i vertikalnom smjeru, te dva operatora za dijagonalne smjerove. Pri tome se u svakoj točki slike označava u kojem smjeru je najveći iznos gradijenta i ta se informacija koristi u kasnijim fazama algoritma.

Najčešće korišteni operator za detekciju rubova jest Sobelov operator kojim se dobiva vrijednost prve derivacije u horizontalnom, te vertikalnom smjeru.

Primjer Sobel operatora:

$$h_1 = \begin{bmatrix} -1 & 0 & 1 \\ -\sqrt{2} & 0 & \sqrt{2} \\ -1 & 0 & 1 \end{bmatrix} \text{ i } h_2 = \begin{bmatrix} -1 & -\sqrt{2} & -1 \\ 0 & 0 & 0 \\ 1 & \sqrt{2} & 1 \end{bmatrix}$$

Formula za izračunavanje gradijenta:

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = arctan\left(\frac{G_y}{G_x}\right)$$

3.1.1.3. Potiskivanje slikovnih elemenata koji nemaju maksimalan gradijent

Sljedećim korakom izvršava se pretraživanje kojim se ispituje lokalni maksimum u smjeru gradijenta. Ovisno o kutu gradijenta, koji predstavlja smjer pojedinog ruba, određuje se je li neka točka na nekom od rubova, odnosno, svaka vrijednost gradijenta koja nije lokalni maksimum postavlja se na nulu. Ovim korakom pronalaze se spojeni skupovi rubnih točaka.

Za svaki slikovni element promatra se matrica 3x3 koja ga okružuje:

- ako je $\theta'(x,y) = 0°$, tada se promatraju i slikovni elementi $(x+1,y)$, (x,y) i $(x-1,y)$, odnosno, slikovni element (x,y) bit će dio ruba ako je njegov intenzitet veći od intenziteta slikovnih elemenata koji su smješteni istočno i zapadno od njega.
- ako je $\theta'(x,y) = 45°$, tada se promatraju i slikovni elementi $(x+1,y+1)$, (x,y) i $(x-1,y-1)$, odnosno, slikovni element (x,y) bit će dio ruba ako je njegov intenzitet veći od intenziteta slikovnih elemenata koji su smješteni sjeveroistočno i jugozapadno od njega.
- ako je $\theta'(x,y) = 90°$, tada se promatraju i slikovni elementi $(x,y+1)$, (x,y) i $(x,y-1)$, odnosno, slikovni element (x,y) bit će dio ruba ako je njegov intenzitet veći od intenziteta slikovnih elemenata koji su smješteni sjeverno i južno od njega.
- ako je $\theta'(x,y) = 135°$, tada se promatraju i slikovni elementi $(x+1,y-1)$, (x,y) i $(x-1,y+1)$, odnosno, slikovni element (x,y) bit će dio ruba ako je njegov intenzitet veći od intenziteta slikovnih elemenata koji su smješteni sjeverozapadno i jugoistočno od njega.

Ako slikovni element (x,y) ima najveći gradijent od preostala tri slikovna elementa, označava se kao da je rub na slici. Ako jedan od ostala dva slikovna elementa ima veći gradijent, tada slikovni element (x,y) nije 'centar'ruba i ne klasificira se kao rub na slici.

3.1.1.4. Histereza pragova

Iako se u prethodna tri koraka nastoji izlučiti što više korisnih informacija, i dalje mogu ostati slikovni elementi koji predstavljaju šum. Histereza razina nastoji eliminirati takve slikovne elemente koristeći dva praga: t_{low} i t_{high}.

Svi slikovni elementi čiji je gradijent niži od t_{low} se odmah eliminiraju. Slikovni elementi čija je vrijednost gradijenta unutar pragova se uzima u obzir jedino ako formiraju kontinuirani rub sa slikovnim elementima s gradijentom koji su veći od t_{high}.

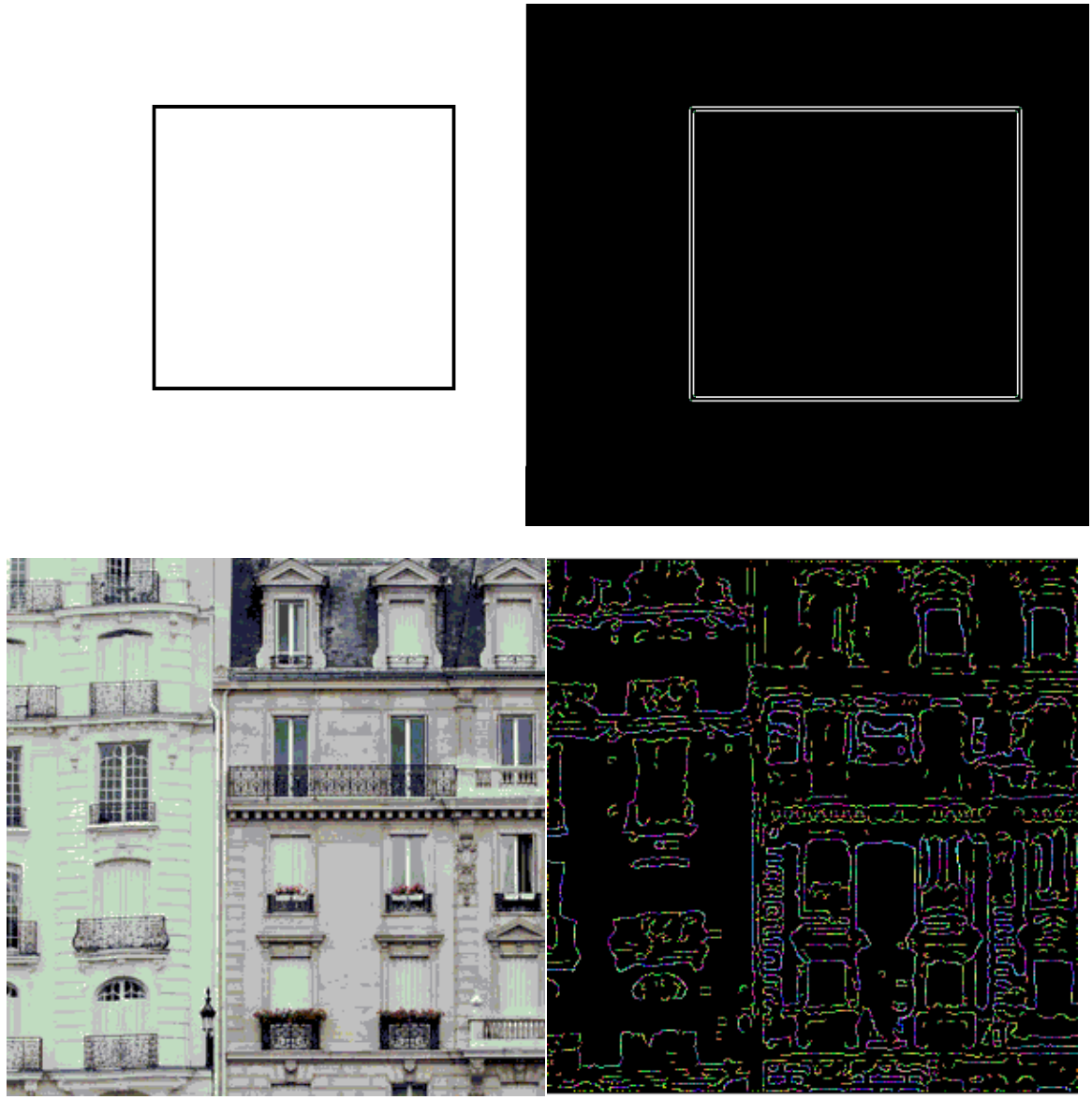
3.2.1. Implementacija Cannyjevog algoritma detekcije rubova

Cannyjev algoritam detekcije rubova korišten je u sklopu posebnog *framework-a* zvanog *Aforge.NET*. *Aforge.NET framework* je *framework* pisan u programskom jeziku C#. Sadrži implementirane algoritme koji se najčešće koriste u područjima računalnog vida i umjetne inteligencije poput obrade slike, neuronskih mreža, strojnog učenja, robotike i ostalih [13].

Klasa *CannyEdgeDetector* kao ulazni podatak dobiva fotografiju čine rubove želimo detektirati te obavlja detekciju rubova pozivanjem metode *Apply* ili *ApplyInPlace*. S obzirom da je jedan od koraka u detekciji rubova već spomenuta histereza pragova, moguće je odrediti i pragove za detekciju čime bi se izdvojili rubovi koji su korisniji od ostalih detektiranih. To se postiže pomoću varijabli *HighThreshold* i *LowThreshold*.

Nedostatak kod korištenja Cannyjevog algoritma iz *Aforge framework-a* jest u tome što klasa *CannyEdgeDetector* kao ulazni podatak zahtijeva fotografije određenog formata slikovnih elemenata, točnije, *8bpp* (8 bitova po slikovnom elementu). Većina snimljenih fotografija je u formatu *32bpp* (32 bita po slikovnom elementu) stoga je potrebno napisati metodu koja će pretvoriti format slikovnih elemenata ulazne fotografije u traženi format. Također, moguće je fotografiju spremiti pomoću nekog od alata za obradu fotografija (npr. Paint) kao *256 Color Bitmap*, no tada će mnogi detalji s fotografije zbog pretvorbe formata fotografije nestati.

Cannyjev algoritam imat će bolje rezultate ako je ulazna fotografija kvalitetnija i ako su rubovi jasniji. Primjer je jasno vidljiv na slici 6. U prvom redu s lijeve strane prikazana je bijela slika s crnim pravokutnikom jasnih i ravnih linija, stoga njen Canny rezultat potpuno točno prikazuje jednostavan rub ulazne fotografije. U redu dolje prikazano je pročelje koje ima mnogo detalja, a ulazna fotografija nije velike razlučivosti (300x300), stoga je njen prikaz Cannyjevog algoritma manje jasan i ima dosta šumova. Također, zbog svijetle boje pročelja lijeve zgrade ne vide se izbočine koje ukrašavaju pročelje, odnosno, algoritam ih ne prepoznaje kao rubove.



Slika 6. Originalne slike prikazane su s lijeve strane, a s desne su prikazani rezultati Cannyjevog algoritma.

3.2. Houghova transformacija

Houghova transformacija omogućuju detekciju rubova objekata na slici. Prije same transformacije potrebno je detektirati rubove na slici, a zatim se kombiniranjem slikovnih elemenata rubnih točaka detektiraju linije i, ako je potrebno, složenije konture.

Kao ulazni podatak za Houghovu transformaciju predaje se slika s detektiranim rubovima. Slika sadrži slikovne elemente koji predstavljaju rub i sikovne elemente koji predstavljaju pozadinu najčešće crne boje.

Kroz svaku točku ruba moguće je provući beskonačno mnogo pravaca

$$y = mx + n.$$

Zadatak Houghove transformacije jest odrediti takve koeficijente m i n da na jednom pravcu leži što više rubnih točaka (slikovnih elemenata). Na taj način transformacija preslikava prostor slike u prostor koji je određen parametrima m i n.

$$HoughT\colon Originalna\ slika(x,y) \rightarrow Parametarski\ prostor(m,n)$$

Svaka rubna točka koja se preslika u parametarski prostor (m,n) daje glas jednom od pravaca. Za svaku rubnu točku originalne slike, (x,y), jednadžba pravaca koji kroz nju prolaze određuje se kao:

$$n = (-x) * m + y.$$

Parametarski prostor još nazivamo i akumulatorskim poljem. Maksimum u akumulatorskom polju predstavlja sjecište u kojem se sječe najviše pravaca i pomoću njega poznati su parametri (m,n) pravca u (x,y) prostoru.

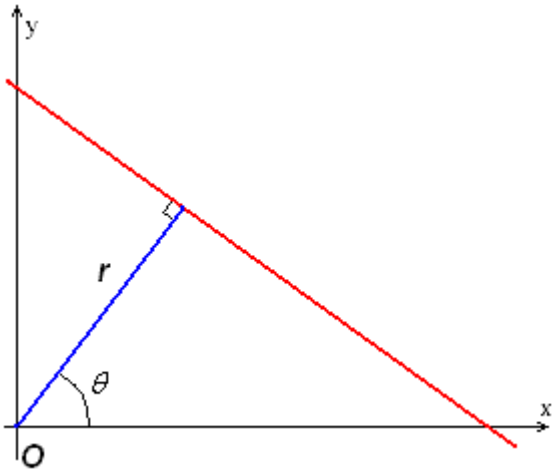
U (m,n) prostoru će uvijek postojati barem jedan maksimum $(\mathrm{m}',\mathrm{n}')$ koji će definirati najveći pravac unutar slike i predstavljati jedan segment u (x,y) prostoru. Maksimumi koji slijede iza maksimuma $(\mathrm{m}',\mathrm{n}')$ definirat će pravce koji sadrže manje točaka, odnosno, manje točaka je glasovalo za njih.

3.2.1. Problem Houghove transformacije

Glavni problem prikaza Houghove transformacije jest u iznosu parametra m koji se može kretati u intervalu $< -\infty, \infty >$. Stoga se, s ciljem izbjegavanja tog problema, najčešće koristi preslikavanje u prostor polarnih koordinata.

$$HoughT: Originalna\ slika(x, y) \rightarrow Parametarski\ prostor\ (\theta, r)$$

Svaki pravac iz (x, y) prostora preslikava se u prostor (θ, r).



Slika 7. Prikaz parametara (θ, r) u xy koordinatnom sustavu

Pravac označen crvenom bojom na slici 4. zapisan je jednadžbom $y = mx + n$. Parametar r koji je na slici označen plavom bojom predstavlja udaljenost između linije i ishodišta koordinatnog xy sustava, a θ je kut vektora od ishodišta do najbliže točke. Koristeći parametrizaciju, jednadžba pravca može se zapisati kao:

$$y = \left(-\frac{\cos\theta}{\sin\theta}\right)x + \left(\frac{r}{\sin\theta}\right)$$

odnosno,

$$r = x\cos\theta + y\sin\theta.$$

Lokalni maksimum $(r_{m,}\theta_m)$ određuje zapis pravca $y = \frac{x\cos(\theta_m) - r_m}{\sin(\theta_m)}$ u (x, y) prostoru.

3.2.2. Implementacija Houghove transformacije

Houghova transformacija također je implementirana pomoću *AForge.NET framework-a.* Klasa *HoughLineTransform* izvodi Houghovu transformaciju nad zadanom fotografijom pomoću metode *ProcessImage*. Pomoću klase *HoughLine* u posebnu listu stavljaju se sve linije koje su kreirane Houghovom transformacijom. Iz liste je zatim moguće izdvojiti samo dio linija s kojima se želi raditi, što su u ovom slučaju sve horizontalne i vertikalne linije. U nastavku je predstavljen dio programskog kôda u programskom jeziku C# koji prikazuje upotrebu klasa *AForge.NET framework-a* u implementaciji Houghove transformacije.

Najprije se izvrši Houghova transformacija nad ulaznom fotografijom a zatim se s obzirom na dobivene linije ispituje kojoj vrsti linija pripadaju: vertikalnim, horizontalnim ili ostalim linijama, što je bitno u nastavku programa.

```
HoughLineTransformation lineTransform = new HoughLineTransformation();
lineTransform.ProcessImage(houghImage);
Bitmap houghLineImage = lineTransform.ToBitmap();
HoughLine[] lines = lineTransform.GetLinesByRelativeIntensity(0.2);

foreach (HoughLine line in lines)
{
      int r = line.Radius;
      double t = line.Theta;

      if (r < 0)
      {
         t += 180;
         r = -r;
      }
            //pretvorba stupnjeva u radijane
      t = (t / 180) * Math.PI;
          // pronaći centre slike
      int w2 = houghImage.Width / 2;
      int h2 = houghImage.Height / 2;

      if (line.Theta == 0 || line.Theta == 180)
      {
           // vertikalna linija
           x0 = line.Radius;
           x1 = line.Radius;

           y0 = h2;
           y1 = -h2;

           horizontal = false;
      }
      else if (line.Theta == 90 || line.Theta == 270)
      {
```

```
                // horizontalna linija
                x0 = -w2;
                x1 = w2;

                y0 = line.Radius;
                y1 = line.Radius;

                horizontal = true;
        }
        else
        {
                 x0 = -w2; // najlijevija točka
                 x1 = w2;  // najdesnija točka

                 y0 = (-Math.Cos(t) * x0 + r) / Math.Sin(t);
                 y1 = (-Math.Cos(t) * x1 + r) / Math.Sin(t);
         }
}
```

Klase koje izvode Houghovu transformaciju također kao ulaznu fotografiju zahtijevaju onu čiji su slikovni elementi formata *8bpp*.

Nakon što se završi Houghova transformacija iscrtavaju se horizontalne i vertikalne pronađene linije što je prikazano na slici 8. Ograničenje kojim se iscrtavaju isključivo horizontalne i vertikalne linije objašnjava se činjenicom da je većina fasada arhitektonskih građevina uglavnom mreža horizontalnih i vertikalnih linija.

Slika 8. S lijeve strane prikazana je ulazna fotografija, a s desne strane prikazane su linije nastale primjenom Houghove transformacije (linije su crvene boje).

3.2.2.1. Sjecišta Houghovih linija

Za potrebe sljedećeg koraka, dekompozicije pročelja, potrebno je odrediti i pohraniti točke u kojima se svaka horizontalna linija siječe sa svakom vertikalnom linijom.

Svaka linija ima svoju početnu i završnu točku. Neka je prva linija omeđena točkama (x_1,y_1) i (x_2,y_2), a druga linija je omeđena točkama (x_3,y_3) i (x_4,y_4).

Sjecište (Tx, Ty) tih dviju linija može se izračunati pomoću determinanti.

$$T_x = \frac{\begin{vmatrix} \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} & \begin{vmatrix} x_1 & 1 \\ x_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & y_3 \\ x_4 & y_4 \end{vmatrix} & \begin{vmatrix} x_3 & 1 \\ x_4 & 1 \end{vmatrix} \end{vmatrix}}{\begin{vmatrix} \begin{vmatrix} x_1 & 1 \\ x_2 & 1 \end{vmatrix} & \begin{vmatrix} y_1 & 1 \\ y_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & 1 \\ x_4 & 1 \end{vmatrix} & \begin{vmatrix} y_3 & 1 \\ y_4 & 1 \end{vmatrix} \end{vmatrix}}$$

$$T_y = \frac{\begin{vmatrix} \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} & \begin{vmatrix} y_1 & 1 \\ y_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & y_3 \\ x_4 & y_4 \end{vmatrix} & \begin{vmatrix} y_3 & 1 \\ y_4 & 1 \end{vmatrix} \end{vmatrix}}{\begin{vmatrix} \begin{vmatrix} x_1 & 1 \\ x_2 & 1 \end{vmatrix} & \begin{vmatrix} y_1 & 1 \\ y_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & 1 \\ x_4 & 1 \end{vmatrix} & \begin{vmatrix} y_3 & 1 \\ y_4 & 1 \end{vmatrix} \end{vmatrix}}$$

4. Dekompozicija pročelja

Dekompozicija pročelja središnji je i najzahtjevniji dio postupka modeliranja urbanih okruženja. Dekompozicijom pročelja nastoji se opisati struktura pročelja, segmentirajući pročelje na najmanji mogući broj elemenata. Tijekom izrade ovog rada isprobano je nekoliko metoda kojima se pokušalo automatski segmentirati elemente pročelja, no sve metode bile su neuspješne i zbog različitih razloga nisu polučile željeni rezultat. Zbog nedostatka vremena nije bilo moguće nastaviti istraživanje vezano uz dekompoziciju podataka, te je odlučeno da će segmentaciju, odnosno odabir elemenata pročelja, obaviti korisnik. U nastavku su objašnjeni pokušaji implementiranja automatskih metoda te konačna metoda, interakcija s korisnikom.

4.1. Dekompozicija temeljena na mreži Houghovih linija i Cannyjevih rubova

U koracima koji prethode dekompoziciji pročelja dobivena je mapa koja sadrži skup svih rubova pronađenih Cannyjevim algoritmom detekcije rubove, te skup svih horizontalnih i vertikalnih linija koje su pronađene Houghovom transformacijom. Horizontalne i vertikalne linije dobivene Houghovom transformacijom kreiraju mrežu linija koje se međusobno sijeku. Sjecišta linija kreiraju velik broj pravokutnika među kojima su i pravokutnici koji predstavljaju elemente pročelja.

Problem koji se javlja u ovoj dekompoziciji jest predetaljna mreža pravokutnika iz koje je potrebno izdvojiti elemente pročelja. Jedino što je poznato iz mreže pravokutnika su mnogobrojna sjecišta linija i pretpostavka da su svi elementi pročelja pravokutnog oblika i da se većina pravokutnih elemenata pročelja vjerojatno ponavlja. Nepoznato je koliko elemenata pročelja je potrebno

pronaći, kao i koje su veličine elementi pročelja te koliko vrsta različitih elemenata pročelja postoji na slici.

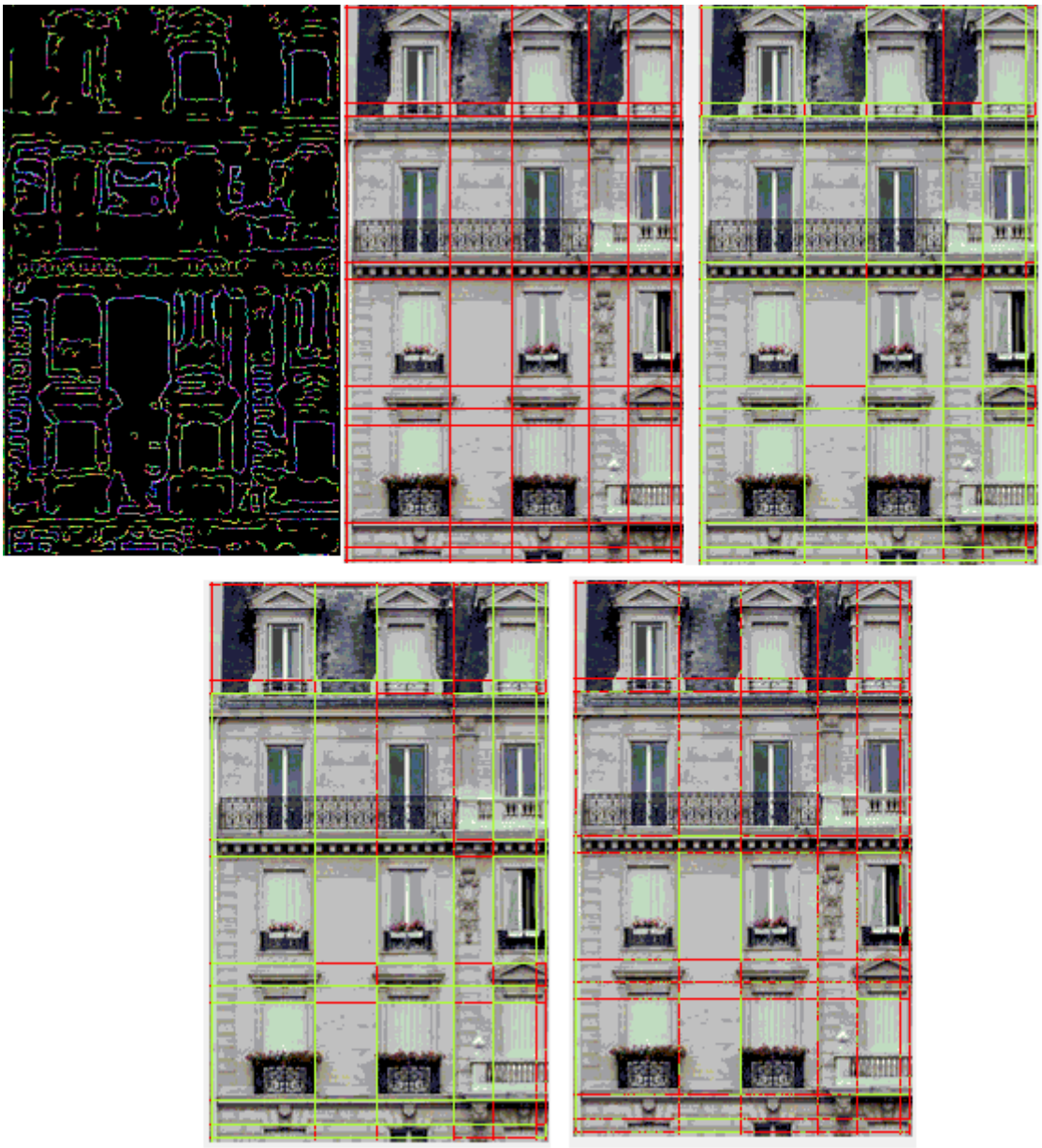
Osim poznatih podataka iz mreže Houghovih linija poznati su i rubovi dobiveni Cannyjevim algoritmom detekcije rubova.

Ideja dekompozicije bila je sljedeća: za svaku horizontalnu liniju koja je omeđena s dva sjecišta ispitaj koliko Canny rubova sadrži – takav podatak možemo zvati *težinom* linije. Nakon što se pregledaju sve linije, spremi one linije (i iscrtaj ih) koje zadovoljavaju određenu težinu. Takve linije dio su elemenata pročelja.

Rezultati dobiveni implementacijom takve dekompozicije nisu se podudarali sa spomenutom idejom. Mijenjanjem parametra kojim se predstavljala granica težine neke linije dobivali su se različiti rezultati, no niti jedan nije bio zadovoljavajući niti iskoristiv za nastavak rada aplikacije.

Ako je kao parametar za granicu težine linije postavljeno da je dovoljno da postoji samo jedan Cannyjev rub koji leži na liniji, kao rezultat se dobivala i dalje pre detaljna mreža pravokutnika. Ako se pak parametar povećao, pa je granica za težinu linije bila 10% od dužine linije, tada bi mreža pravokutnika bila nedovoljno precizna, jer je bilo malo pravokutnika koji su se podudarali s pravim elementima pročelja. S daljnjim povećanjem parametra granice težine smanjivala se preciznost mreže pravokutnika i bila je posve neupotrebljiva, što je vidljivo na slici 9.

Dakako, odabir parametara nije jedini razlog zbog kojeg se nije mogao ostvariti dobar rezultat. Ako je ulazna fotografija već u početku bila nedovoljno detaljna, i njene obrade (Cannyjevi rubovi i Houghove linije) bit će nedovoljno detaljne, što znači da će i kompozicija obrada biti nedovoljna i nepotpuna.



Slika 9. Ovisnost rezultata dekompozicije s obzirom na parametar težine linije. Na prvoj slici prikazani su Cannyjevi rubovi, zatim linije Houghove transformacije. Treća slika u prvom redu zelenom bojom prikazuje linije koje sadrže barem jedan Cannyjev rub. U drugom redu prikazane su linije koje sadrže 10%, odnosno, 50% Cannyjevih rubova.

4.2. Dekompozicija uz *FloodFill* agoritam

Sljedeći pokušaj dekompozicije odvija se pomoću već postojeće mreže Houghovih linija i Cannyjevih rubova, ali i uz korištenje *FloodFill* algoritma.

4.2.1. *FloodFill* algoritam

FloodFill algoritam jedan je od poznatih algoritama koji se koristi u aplikacijama poput *Paint*-a kao jedna od opcija za popunjavanje dijela slike odabranom bojom, gdje je predstavljen ikonom kantice s bojom. Također se koristi i u računalnim igrama poput *Go*-a i *Minesweeper*-a za određivanje područja koji su prazni. Još se naziva i *seed fill* ili *boundary fill* algoritmom.

FloodFill algoritam koristi tri parametra: početni čvor, tražena boja i zamjenska boja. Njegov zadatak je pretražiti sve čvorove u listi ili polju koji su povezani s početnim čvorom i obojani traženom bojom, te ih obojiti u zamjensku boju. Algoritam može pretraživati čvorove u četiri smjera (istok, zapad, sjever, jug) ili osam smjerova (istok, zapad, sjever, jug, sjeveroistok, sjeverozapad, jugoistok, jugozapad). Postoji nekoliko verzija *FloodFill* algoritma, a u sklopu ovog rada isprobane su dvije verzije.

Najjednostavnija verzija, ali ujedno i najmanje optimalna verzija, jest klasični rekurzivni *FloodFill* algoritam.

Pseudokôd rekurzivnog *FloodFill* algoritma je sljedeći:

```
Flood-fill (cvor, trazena-boja, zamjenska-boja):
  Ako boja cvora nije jednaka trazenoj-boji, vrati se.
  Postavi boju cvora na zamjensku-boju.
  Izvedi Flood-fill (zapadni-cvor, trazena-boja, zamjenska-boja).
  Izvedi Flood-fill (istocni-cvor, trazena-boja, zamjenska-boja).
  Izvedi Flood-fill (sjeverni-cvor, trazena-boja, zamjenska-boja).
  Izvedi Flood-fill (juzni-cvor, trazena-boja, zamjenska-boja).
  Vrati se.
```

Nedostatak rekurzivnog FloodFill algoritma je u duljini trajanja pretraživanja i rekurzije, ali i brzog zasićenja memorije zbog korištenja stoga, što je često neučinkovito kod većine programskih jezika.

Sljedeća implementirana verzija FloodFill algoritma ne koristi stog već red (engl. *queue*) pa ne dolazi do zasićenja memorije, no nije brži od klasičnog rekurzivnog algoritma.

Pseudokôd je sljedeći:

```
Flood-fill (cvor, trazena-boja, zamjenska-boja):
  Postaviti Q kao prazan red.
  Ako boja cvora nije jednaka trazenoj-boji, vrati se.
  Dodaj cvor na kraj reda Q.
  Dok Q nije prazan:
      Postavi n jednak prvom elementu reda Q
      Ako je boja n-a jednaka trazenoj-boji, postavi boju n-a u
zamjensku-boju.
      Makni prvi element iz reda Q
      Ako je boja zapadnog-cvora od cvora n je trazena-boja:
          Dodaj cvor na kraj reda Q.
      Ako je boja istocnog-cvora od cvora n je trazena-boja:
         Dodaj cvor na kraj reda Q.
      Ako je boja sjevernog-cvora od cvora n je trazena-boja:
         Dodaj cvor na kraj reda Q.
      Ako je boja juznog-cvora od cvora n je trazena-boja:
         Dodaj cvor na kraj reda Q.
 Vrati se.
```

4.2.2. Implementacija *FloodFill* algoritma

Ideja dekompozicije koristeći *FloodFill* algoritam je da se rekurzivno pretražuje mreža pravokutnika i pritom boji različitim bojama (ovisno o dubini rekurzije) čime bi se ne samo smanjila detaljnost mreža pravokutnika već i stvorila mapa slična dubinskoj mapi.

Čvor koji je ulazio kao početni parametar *FloodFill* algoritma jest slikovni element koji dolazi nakon prvog sjecišta prve horizontalne linije, tražena boja je boja kojom su obojane linije (crvena), a zamjenska boja je jedna od nijansi sive boje. S obzirom na dubinu rekurzije, to je nijansa sive boje tamnija, a element pročelja veće dubine.

Obje verzije *FloodFill* algoritma minimalno su izmijenjene s obzirom na zadatak koji im je zadan. Kad se pregledava boja čvora, odnosno, slikovnog elementa, pregledava se je li različita od tražene boje, a ne jednaka kako je zapisano u originalnom algoritmu. Razlog tomu je što cilj našeg zadatka nije zamjenskom bojom popuniti područje koje je obojeno traženom bojom, već zamjenskom bojom popuniti područje koje je omeđeno traženom bojom, dakle, područje koje nije jednako traženoj boji.

Problem kod korištenja rekurzivne verzije *FloodFill* algoritma jest dugotrajno izvođenje algoritma uz vraćanju obavijesti o popunjenom stogu i nedostatku memorije. Kod korištenja verzije *FloodFill* algoritma uz korištenje reda nije bilo iznimaka o nedostatku memorije, ali su se javile iznimke o stajanju dretve, odnosno, da dretva koja treba obavljati *FloodFill* zadatak predugo ne radi.

Kako rješenja ovih problema nisu nađena, odlučeno je da će dekompoziciju pročelja obavljati korisnik.

4.2.3. Dekompozicija uz interakciju korisnika

Nakon što se korisniku prikaže fotografija pročelja, potrebno je, korištenjem računalnog miša, označiti pravokutnike na onim mjestima na slici gdje pročelje treba biti izbočeno ili udubljeno.

Pokazivač računalnog miša potrebno je postaviti na onaj slikovni element kojeg smatramo gornjim lijevim vrhom pravokutnika koji će predstavljati jedan od elemenata pročelja. Zatim je potrebno pritisnuti lijevu tipku računalnog miša i povući pokazivač miša sve do donjeg desnog vrha željenog pravokutnika. Označeni pravokutnik će se obojiti određenom bojom i prikazati zajedno s fotografijom, kao što je prikazano na slici 10.

Tijekom označavanja pravokutnika, sustav će pamtiti koordinate pojedinih pravokutnika i spremati ih u zasebne instance klasa pravokutnika koje se zatim spremaju u listu svih pravokutnika fotografije.

Slika 10. Dekompozicija uz interakciju korisnika. Ljubičasti pravokutnici označavaju elemente pročelja. Manji pravokutnici predstavljaju prozore, a veliki pravokutnik ulazna vrata.

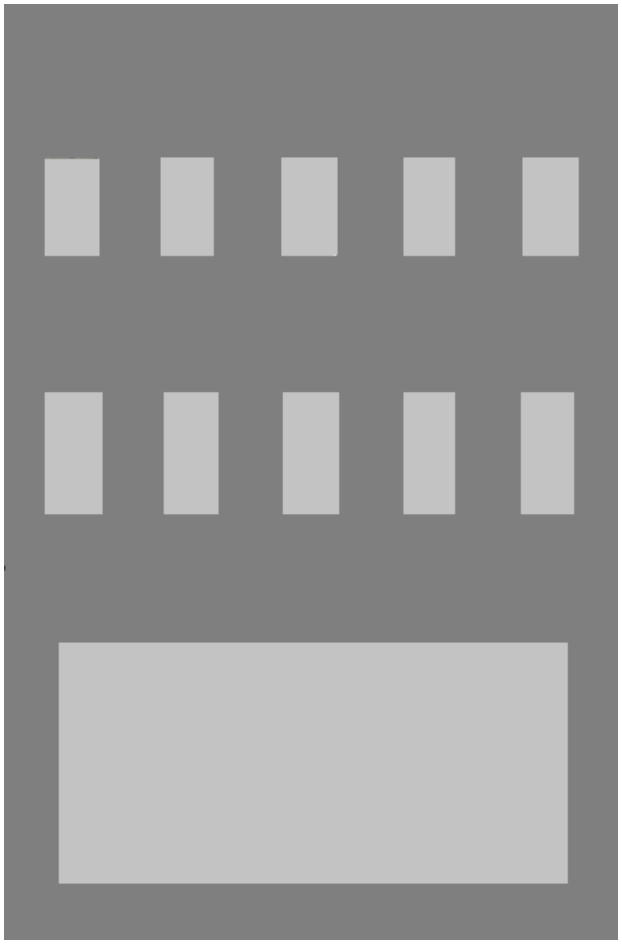
5. Kreiranje dubinske mape

Dubinska mapa (engl. *depth map*) je slika visine i širine kao i originalna fotografija, a sadrži elemente pročelja određene dekompozicijom. Svaki element pročelja u dubinskoj mapi obojen je određenom nijansom sive boje, s obzirom na dubinu koju označava. Elementi pročelja koji su jednake dubine na dubinskoj mapi prikazani su jednakom bojom.

S obzirom da je dekompozicija pročelja prestala biti automatizirana i obavlja se interakcijom korisnika, dubinska mapa nije niti potrebna jer se svi podaci koji su potrebni u zadnjem koraku izrade modela nalaze u listi pravokutnika. Ipak, implementacija stvaranja dubinske mape bit će zadržana, ukoliko se nastavi raditi na ovom radu i dekompozicija se automatizira.

Ukoliko se dekompozicija automatizira, dobiveni elementi pročelja bit će predstavljeni na dubinskoj mapi različitim bojama. Svaka boja predstavljat će određenu dubinu i korisnik će prije izrade 3D modela moći ispraviti moguće greške nastale prilikom automatizacije.

Na slici 11. prikazan je primjer dubinske mape.



Slika 11. Dubinska mapa nastala nakon dekompozicije prikazane na slici 10.

6. Izrada 3D modela

Posljednji zadatak u implementaciji je izrada 3D modela i omogućavanje korisniku kretanje po sceni.

Izrada 3D modela omogućena je korištenjem *Open Toolkit (OpenTK)* biblioteke, biblioteke pisane u programskom jeziku C# i koja obuhvaća OpenGL, OpenCL i OpenAL. Često se koristi za računalne igre te ostale aplikacije i projekte koji koriste računalnu grafiku.

Podaci koji su potrebni za izradu modela i scene zapisani su u listi svih pravokutnika koji predstavljaju elemente pročelja. Lista se nakon dekompozicije pročelja i izrade dubinske mape predaje metodama OpenGL klase.

Ideja je izraditi onoliko 3D modela koliko je fotografija korisnik predao sustavu na početku rada aplikacije, pod pretpostavkom da jedna fotografija predstavlja jednu građevinu.

Najprije se kreira jednostavan kvadar visine i širine jednak visini i širini ulazne fotografije i na njega se dodaje tekstura, odnosno ulazna fotografija građevine čiji se model izgradio. Zatim se izgrađuje ostatak pročelja, točnije, elementi pročelja pohranjeni u listi svih elemenata. Za svaki element pročelja poznati su sljedeći podaci: visina, širina i dubina elementa, te koordinate sva četiri vrha koji pripadaju pravokutniku. Iako su poznata sva četiri vrha, nije ih potrebno poznavati sve, već samo gornji lijevi vrh iz kojeg je zatim jednostavno odrediti preostala tri vrha. Također, poznavajući koordinate vrhova lako je dobiti teksturu za element pročelja tako što se kopiraju svi slikovni elementi koji su omeđeni poznatim vrhovima i pohrane u novu sliku.

Nakon što se kreiraju modeli i na njih dodaju teksture, korisnik može bolje pregledati scenu koristeći se tipkama W, A, S i D, koje omogućavaju kretanje u svim smjerovima.

Kretanjem po sceni mijenja se položaj kamere koja prikazuje scenu. Svakom promjenom u sceni ponovno se iscrtava scena u prozoru stoga je

potrebno dobro organizirati redoslijed operacija prilikom iscrtavanja scena kako kretanje po sceni ne bi bilo otežano i usporeno.

Što više modela i tekstura se treba iscrtati, to je iscrtavanje scene sporije, pogotovo ako za svaki model postoji i tekstura koja se treba iscrtati.

Jedno od rješenja takvog problema jest učitati sve teksture na početku iscrtavanja scene, a zatim prije iscrtavanja svakog od modela metodi javiti koju teksturu treba iscrtati. Time se izbjegava nepotreban proces učitavanja teksture svakim pomakom kamere u sceni. Svaka tekstura koja prolazi proces učitavanja dobiva poseban identifikacijski broj teksture koji se predaje metodi za icrtavanje.

Programski kôd za metodu učitavanja tekstura je sljedeći:

```
private int LoadTexture(string path, int id)
{
      Bitmap bmp = new Bitmap(path);
      BitmapData bmp_data = bmp.LockBits(
         new Rectangle(0, 0, bmp.Width, bmp.Height),
         ImageLockMode.ReadOnly,
         System.Drawing.Imaging.PixelFormat.Format32bppArgb);
      id = GL.GenTexture();

      GL.BindTexture(TextureTarget.Texture2D, id);

      GL.TexImage2D(
         TextureTarget.Texture2D,
         0,
         PixelInternalFormat.Rgba,
         bmp_data.Width,
         bmp_data.Height,
         0,
         OpenTK.Graphics.OpenGL.PixelFormat.Bgra,
         PixelType.UnsignedByte,
         bmp_data.Scan0);
      GL.Finish();

      bmp.UnlockBits(bmp_data);

      GL.TexParameter(
         TextureTarget.Texture2D,
         TextureParameterName.TextureMinFilter,
         (int)TextureMinFilter.Linear);
      GL.TexParameter(
         TextureTarget.Texture2D,
         TextureParameterName.TextureMagFilter,
      (int)TextureMagFilter.Linear);

      GL.Enable(EnableCap.Texture2D);

      return id;
}
```

Nakon što se svaku teksturu učita i pohrani njen identifikacijski broj, prije iscrtavanja scene potrebano je pozvati metodu *BindTexture*:

```
GL.BindTexture(TextureTarget.Texture2D, id);.
```

Model, odnosno kvadar, iscrtava se stranu po stranu pozivanjem sljedeće metode za svaki vrh pravokutnika koji predstavlja pojedinu stranu modela:

```
GL.Vertex3((float) x, (float) y, (float) z);
```

Tekstura se na model dodaje na način da se svaki vrh pravokutnika spoji s jednim vrhom teksture:

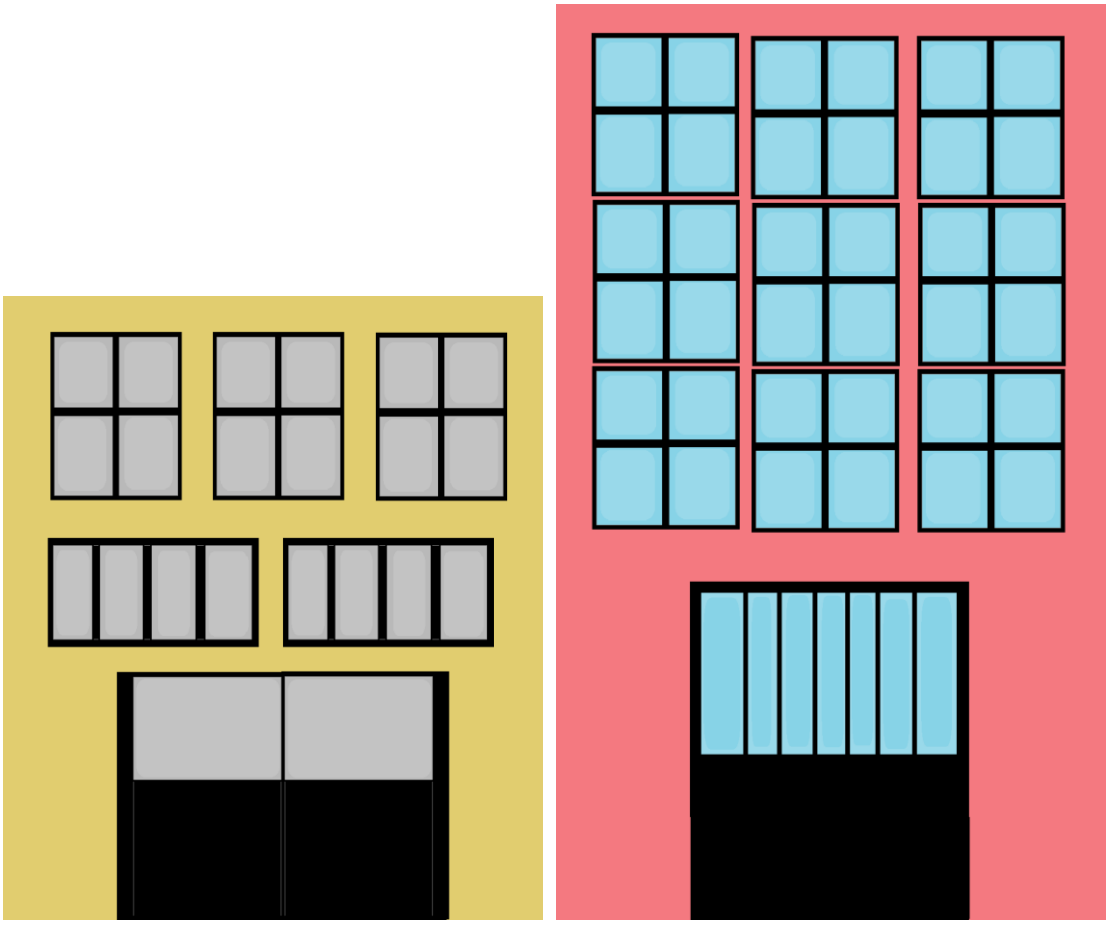
```
GL.TexCoord2((float) x_tex, (float) y_tex);
GL.Vertex2((float) x, (float) y);
```

7. Rezultati

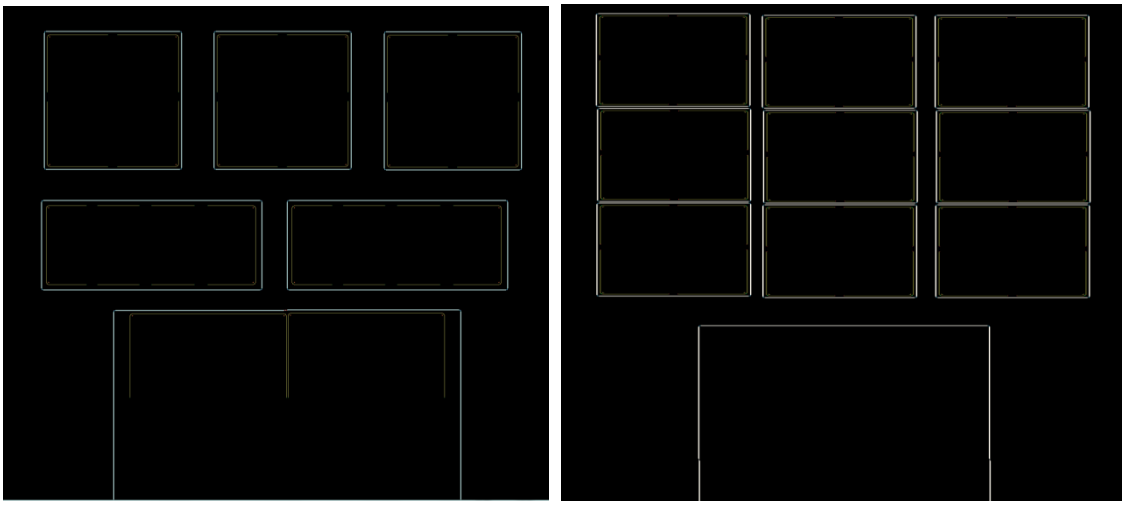
U nastavku je prikazana izrada modela i scene za dva primjera.

Prvi primjer su dvije slike izrađene u *Windows* aplikaciji *Paint.* Namjerno su odabrane ručno obrađene slike kako bi se vidjeli najbolji rezultati jer se radi o slikama koje sadrže samo horizontalne i vertikalne linije. Fotografije koje su snimljene u realnom svijetu daju puno lošije rezultate jer se većinu linija ne može detektirati (nisu savršeno horizontalne niti vertikalne).

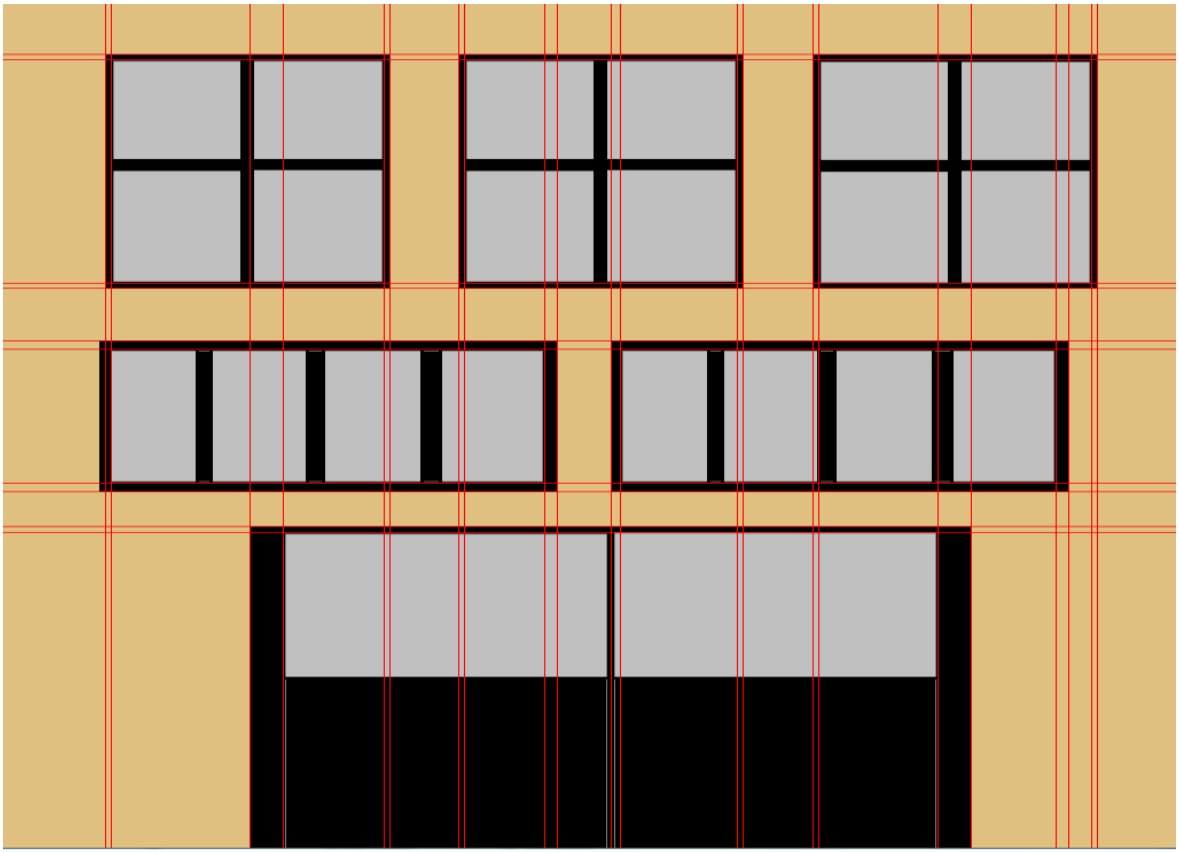
Ulazne fotografije:

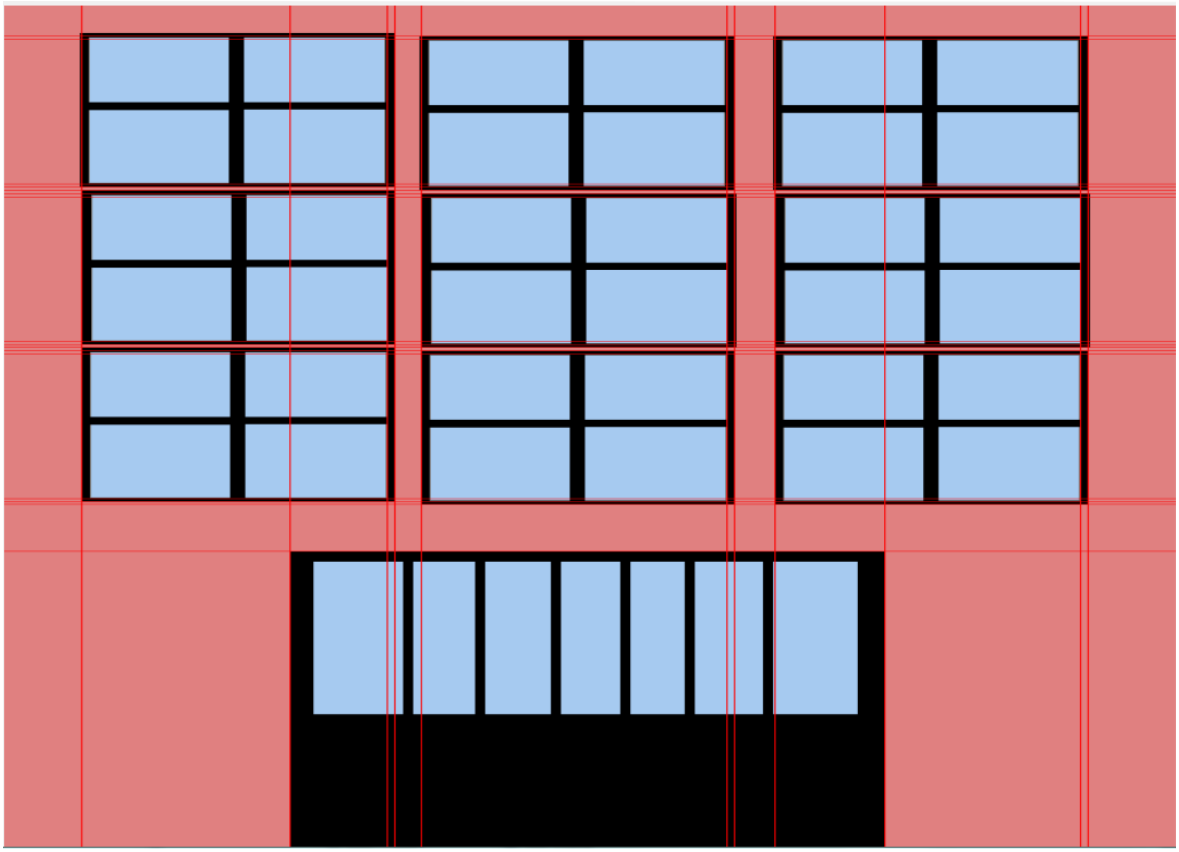


Cannyjeve mape:

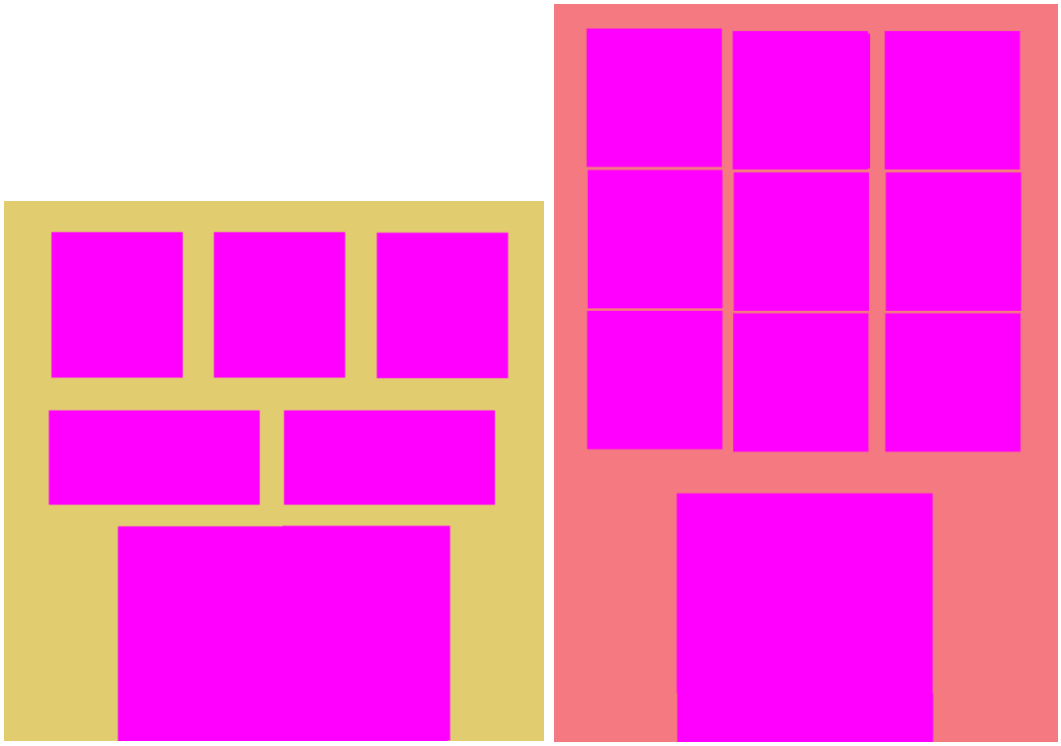


Houghova transformacija:

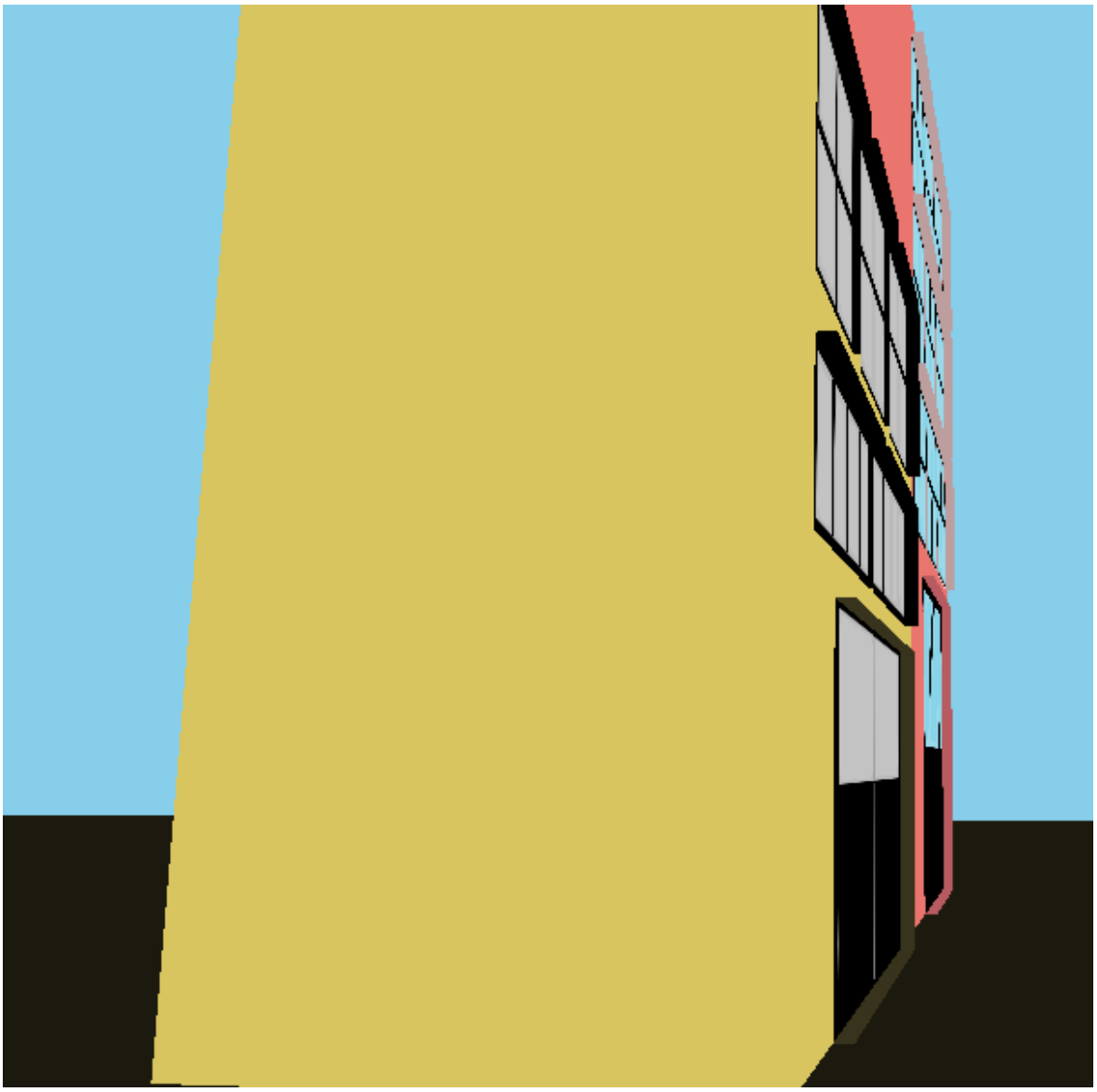




Dekompozicija pročelja:

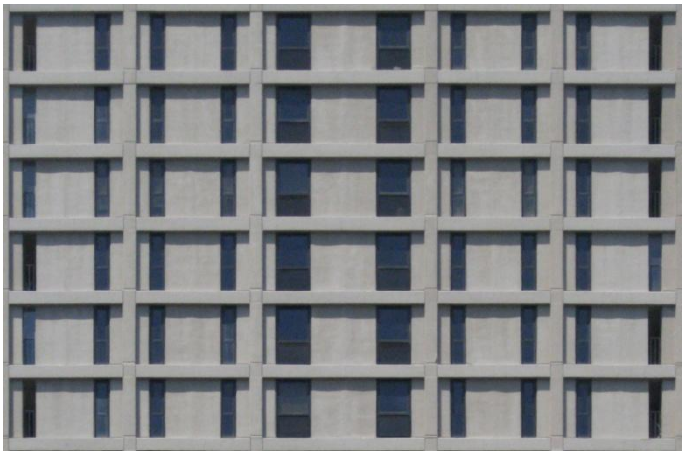


3D model:

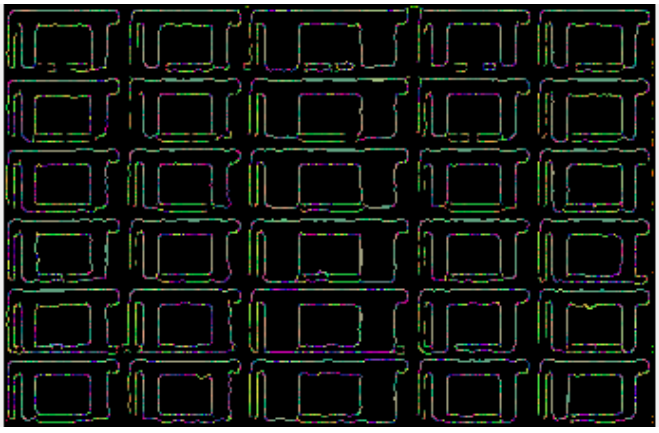


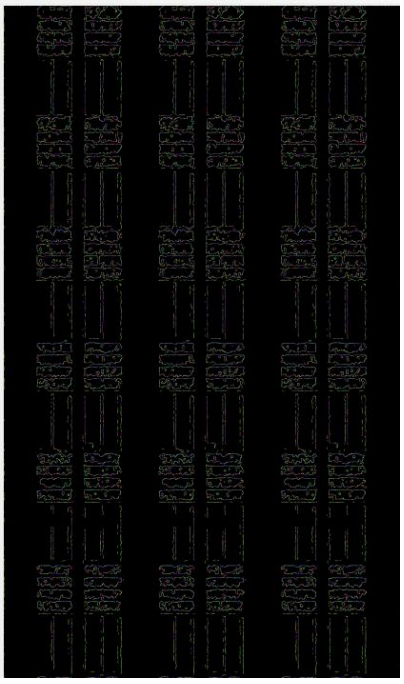
Drugi primjer su dvije fotografije pročelja preuzete s Interneta na kojima se također može dobro detektirati horizontalne i vertikalne linije.

Ulazne fotografije:



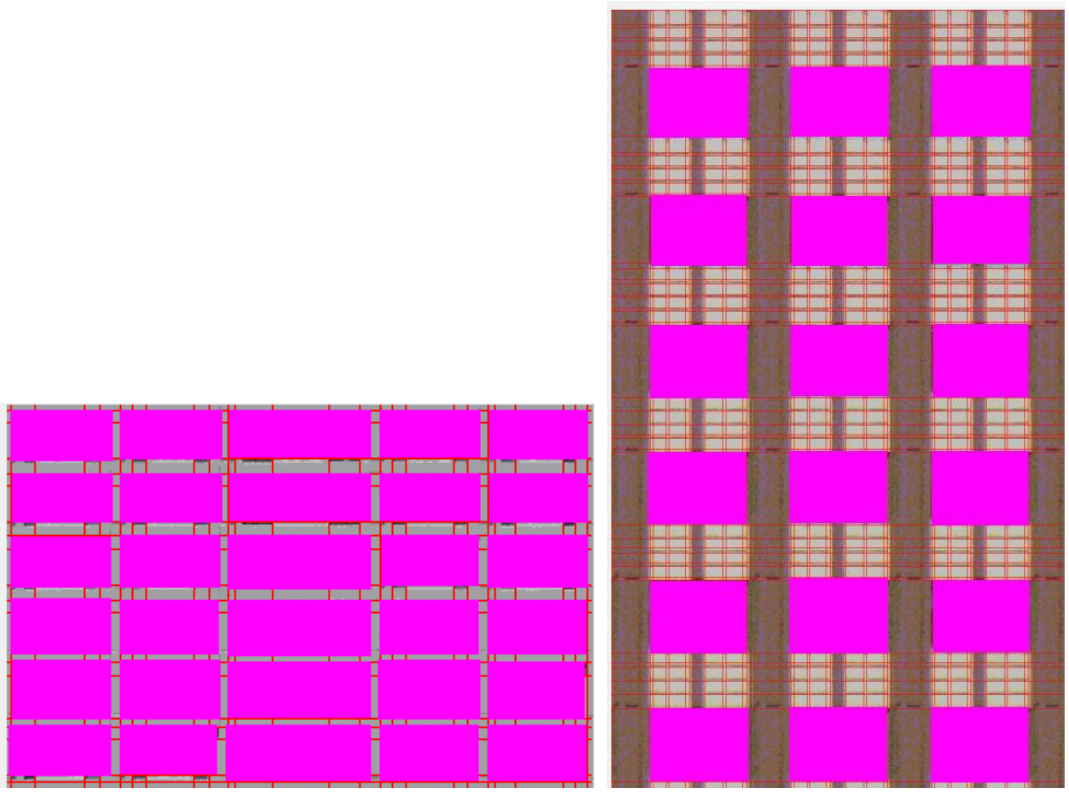
Cannyeve mape:





Houghova transformacija:

Dekompozicija pročelja:



3D model:

8. Zaključak

Modeliranje urbanih okruženja nije jednostavan problem. Potrebno je primijeniti mnogo znanja iz računalne grafike, računalnog vida i obrade slike kako bi se postigli zadovoljavajući rezultati, pogotovo ako se želi postići rezultat koji što više podsjeća na originalna urbana okruženja. Metode koje su dosad osmišljene, iako ostvaruju dosta dobre rezultate, i dalje treba poboljšavati i optimizirati.

Metoda koju se nastojalo ostvariti u sklopu ovog diplomskog rada pokazala se kao zanimljiva, ali i zahtjevna, te je pojedine dijelove postupka modeliranja svakako potrebno još proučiti, nadograditi i poboljšati. Prostor za poboljšanje je svakako velik, no najviše truda trebalo bi se uložiti u dekompoziciju pročelja, koja je jedan od bitnijih i izazovnijih koraka modeliranja pročelja.

Literatura

[1] QUAN, L. Image-Based Modeling. Springer, 2010.

[2] MÜLLER, P., WONKA, P., HAEGLER, S., ULMER, A., AND VAN GOOL, L. 2006. Procedural Modeling of Buildings. In *Proceedings of ACM SIGGRAPH 2006 / ACM Transactions on Graphics*, ACM Press, New York, NY, USA, vol. 25, 614-623.

[3] FRÜH, C., AND ZAKHOR, A. 2003. Constructing 3d city models by merging ground-based and airborne views. *Computer Graphics and Applications* (Nov.), 52-61

[4] DEBEVEC, P. E., TAYLOR, C. J., AND MALIK, J. 1996. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *Proceedings of ACM SIGGRAPH 96*, ACM Press, H. Rushmeier, Ed., 11-20.

[5] CANNY, J. 1986. A computational approach to edge detection. I*EEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698.

[6] ROODT, Y., VISSER, W. AND CLARKE, W. A. Image Processing on the GPU: Implementing the Canny Edge Detection Algorithm.

[7] CRIMINISI, A., PEREZ, P. AND TOYAMA, K. 2003. Object removal by exemplar-based inpainting. In *Proceedings of IEEE Computer Vision and Pattern Recognition*, volume 2, pages 721–728, Madison, Wisconsin.

[8] OH, B. M., CHEN, M., DORSEY, J. AND DURAND, F. 2001. Image-based modeling and photo editing. In Eugene Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 433–442. ACM Press / ACM SIGGRAPH.

[9] MÜLLER, P., ZENG, G., WONKA, P. AND VAN GOOL, L. 2007. Image-based procedural modeling of façades. *Proceeding of SIGGRAPH conference*, 26:85.

[10] WERNER, T. AND ZISSERMAN, A. New Techniques for Automated Architectural Reconstruction from Photographs

[11] DUDA, R. O. AND HART, P. E. 1971. Use of the Hough transformation to detect lines and curves in pictures. *Communications of the Association for Computing Machinery*, 15:11–15, 1972.

[12] CORNELIS, N., LEIBE, B., CORNELIS, K. AND VAN GOOL, L. 2008. 3D urban scene modeling integrating recognition and reconstruction. *International Journal of Computer Vision*, 78(2):121–141.

[13] *AForge.NET Framework*, www.aforgenet.com/framework/, 15.4.2011.

[14] *The Open Toolkit Library*, www.opentk.com, 2.5.2011.

[15] *MSDN Library*, http://msdn.microsoft.com/en-us/library/ms123401.aspx, 10.4.2011.

Sažetak

Naslov: Postupci modeliranja pročelja zgrada temeljenih na fotografijama

Sažetak:

Rastom tržišta računalnih igara i filmova, te ostalih sadržaja vezanih uz računalnu grafiku, raste i potražnja za što realnijim prikazom 3D modela. Ovim radom nastoji se predstaviti najpoznatije metode modeliranja urbanih okruženja te implementirati metodu temeljenu na fotografijama. Proces započinje obradom ulaznih fotografija koja kao rezultat daje mapu detektiranih rubova na fotografijama te prikaz horizontalnih i vertikalnih linija koji kreiraju mapu pravokutnika među kojima se nalaze i elementi pročelja. Dekompozicijom pročelja, koja se odvija interakcijom korisnika, dobivaju se elementi pročelja koje se zatim, zajedno s ostatkom zgrade, iscrtava u 3D sceni.

Ključne riječi: modeliranje pročelja zgrada, računalna grafika, računalni vid, obrada slike, Cannyjev algoritam detekcije rubova, Houghova transformacija, dekompozicija pročelja, FloodFill algoritam, 3D model, dubinska mapa

Summary

Title: Image-based façade building modeling methods

Summary:

The market of computer games, movies and other computer graphics projects is growing rapidly from year to year, and so is the request for more realistic 3D modeling. With this work we try to introduce the most famous façade building modeling methods and implement the image-based façade building modeling method. The method begins with processing input images that gives edge detector map and rectangle map that consists of horizontal and vertical lines. Within that map are the elements of the façade that we try to extract during the decomposition of the façade. The decomposition is interfaced by user and it gives us façade elements that, with the rest of the building, render in 3D scene.

Key words: façade building modeling, computer graphics, computer vision, image processing, Canny Edge Detector Algorithm, Hough Transform, façade decomposition, FloodFill Algorithm, 3D modeling, depth map.