SADRŽAJ

1. PREDGOVOR

Zahvaljujem mentoru red. prof. dr. sc. Zoranu Mrši koji je pratio cijeli proces nastajanja diplomskog rada i svojim savjetima i entuzijazmom usmjeravao me kako da prevladam probleme koji bi se pojavili prilikom izrade diplomskog rada. Hvala što mi je omogućio korištenje računala i odgovarajućeg software-a na svom zavodu bez kojeg ne bi mogao realizirati svoj rad. Želim spomenuti i nesebičnu pomoć asistenta Marka Čavraka, dipl. ing. koji je pomogao svojim iskustvom rada na sličnom problemu.

2. UVOD

Diplomski rad ″Simulacija dinamike perjanice sumpornog dioksida pri nestacionarnim meteorološkim uvjetima″ treba pokazati disperziju sumpornog dioksida pri nestacionarnim uvjetima na modelu terena Bakarskog zaljeva i njegove bliže okolice. Svrha poznavanja pojava disperzije sumpornog dioksida u atmosferi značajna je radi određivanja koncentracija onečišćenja koja utječu na zagađenje okoliša na tom području.

Izvor sumpornog dioksida su dva dimnjaka industrijskog postrojenja INA-e na Urinju. Za trodimenzionalni model terena i generiranje odgovarajuće mreže nad njim korišten je računalni program I-DEAS. Simulacija disperzije sumpornog dioksida napravljena je u računalnom programu FLUENT. Simulacija je izvedena na dva načina. U prvom slučaju rubni uvjeti dobiveni su na osnovu sinusne funkcije koju sam definirao. U drugom slučaju rubni uvjeti potrebni za modeliranje u FLUENT-u dobiveni su iz meteorološkog programa MM5. Na kraju, dan je prikaz dobivenih rezultata, te njihovo pojašnjenje.

Odabir ove teme logičan je nastavak dosadašnjeg rada na kolegiju Računarska dinamika fluida.

3. KREIRANJE TRODIMENZIONALNOG MODELA TERENA

Kod svakog numeričkog modeliranja, dvodimenzionalnog ili trodimenzionalnog, potrebno je definirati zatvorenu domenu unutar koje će se vršiti modeliranje. Za modeliranje difuzije i konvekcije onečišćenja iz dimnjaka na Urinju korišten je kutijasti model čije su stranice određene donjom granicom koju predstavlja topografija terena te gornjom granicom koju predstavlja xy ploha na visini od 2000 m nadmorske visine. Visina od 2000 metara odabrana je kako ne bi utjecala na vertikalnu disperziju na način da ograniči širenje perjanice dima. Bočne stranice modela definirane su krajevima korištene topografije terena gdje razlikujemo istočnu stranicu (smjer Kraljevica), zapadnu stranicu (smjer Kostrena), južnu stranicu (smjer mora) te sjevernu stranicu (smjer Gorski Kotar) (Slika 3.1.). U tako definiran kutijasti model dodana su dva dimnjaka iz postrojenja rafinerije nafte u Urinju iz kojih ćemo pratiti emisije onečišćenja u atmosferu. Lokacije te dimenzije dimnjaka dane su sljedećim podacima:

Inicijalne koordinate

Xo = 5459500.0
Yo = 5010000.0

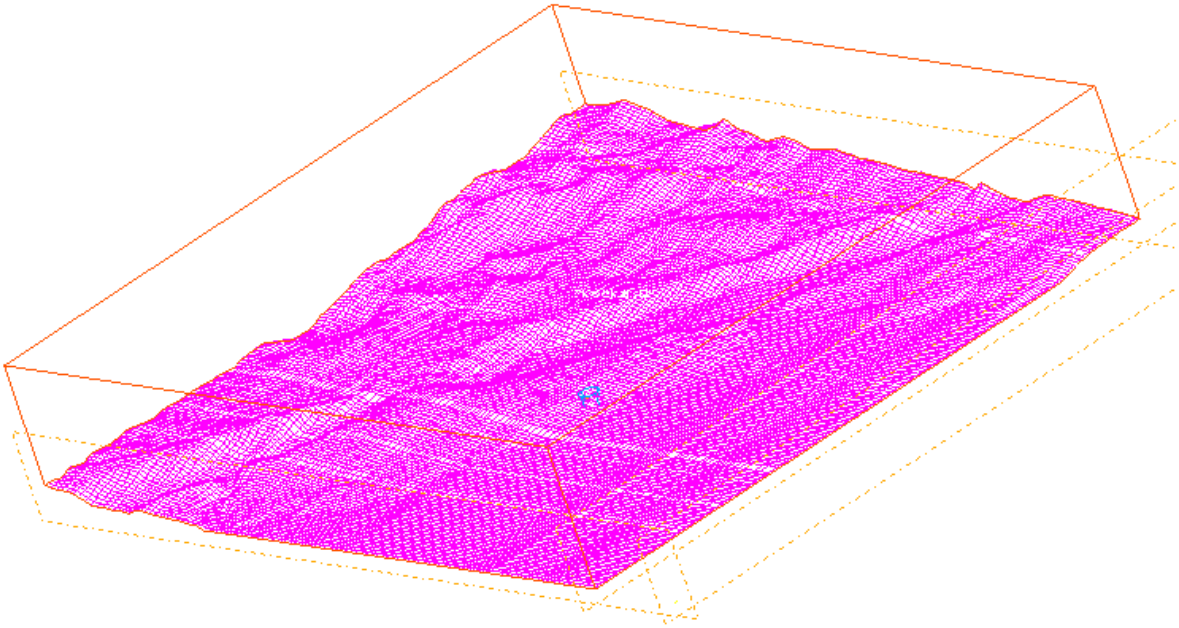
Tablica 3.1.

Zajednički dimnjak	Apsolutne koordinate			Relativne koordinate			Visina dimnjaka	Promjer dimnjaka
	Xa (m)	Ya (m)	Za (m)	Xr (m)	Yr (m)	Zr (m)	Hs (m)	Ds (m)
G4+G5	5463864.40	5015393.00	7.70	4364.40	5393.00	7.70	35.00	2.62
FCC	5463400.00	5015900.00	28.00	3900.00	5900.00	28.00	124.00	4.10

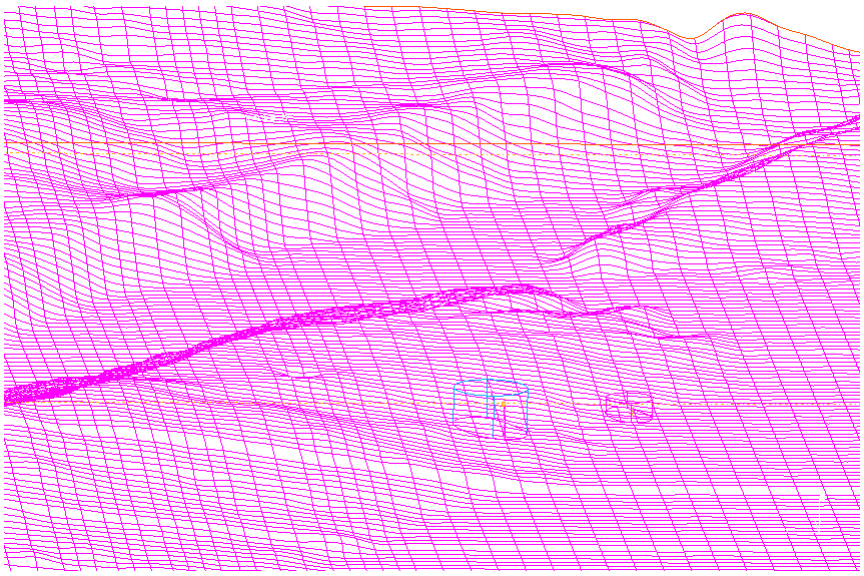
Za definiciju topografije terena korišteni su podaci o receptorskim visinama za svakih sto metara udaljenosti u smjeru x i y osi kartezijevog koordinatnog sustava. Ukupan broj točaka iznosi u x smjeru 113 te u y smjeru 121 što ukupno daje 13673 točaka za koje su poznate njihove visine u smjeru z osi. Ti podaci primarno su dani u tekstualnoj datoteci

složeni u jednostavnom formatu. Budući da smo odabrali I-DEAS CAD program za generiranje geometrije modela, na osnovu ovih podataka potrebno je bilo konvertirati podatke o koordinatama točaka u jedan od formata kojim bi I-DEAS mogao učitati zadane točke terena (konverzija u C++-u). Nakon konverzije formata iz tekstualnog zapisa u I-DEAS-u primjeren Iges format, točke smo importirali u IDEAS. Za formiranje plohe terena koristili smo I-DEAS-ovu naredbu za generiranje plohe kroz matricu točaka (*Fit Surface to Points*). Paralelno tome definiran je kvadar čiju osnovicu čini površina projekcije plohe terena na XY plohu sa koordinatom Z = 0, a visinu predstavlja unaprijed definirana vrijednost od 2000 m. Iz donje strane kvadra izrezali smo plohu terena te dobili kvadar čiju je donju stranicu zamijenila ploha terena. Na taj način formiran je početni kutijasti model (Slika 3.1.). Nadalje, kako bismo definirali i u kutijasti model unijeli dva dimnjaka nacrtali smo dva cilindra dužine i promjera prema podacima o dimnjacima iz Tablice 3.1. Na temelju podataka o lokaciji dimnjaka u relativnim koordinatama odrezali ta dva dimnjaka iz kutijastog modela upravo na prave lokacije u ovom relativnom prostoru (Slika 3.2.).

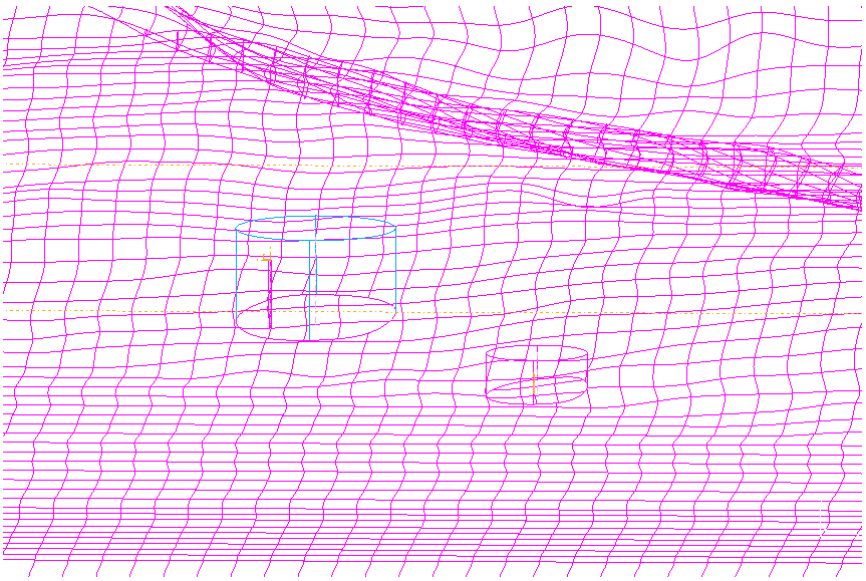
Sljedeći korak je generiranje mreže. Kako bi generirali mrežu nad kutijastim modelom podignut je I-DEAS-ov modul za simulaciju *Simulation* te pokrenuli ploču s alatom za umrežavanje, *Meshing*. Odabran je trokutasti tip elemenata za umreženje rubnih stranica, terena te gornje granice (*Shell Meshing*). Za generiranje mreže unutar modela (*Solid Mesh*) odabran je tetraedarski tip elementa. Za vrstu mreže odabrana je nestrukturirana mreža (*free meshing*) kako bi lakše prionula na plohu terena. Odabir broja elemenata kreće se od 2 do 5 elemenata na plaštu i vrhu dimnjaka, 100 na terenu, 500 na rubnim stranicama te 1000 na gornjoj stranici. Nakon provedenog umreženja, a zbog činjenice da su dimnjaci, geometrijski gledano, mali u usporedbi sa cijelom domenom uočeno je najgušće umreženje oko dimnjaka te u pojedinim djelovima plohe terena gdje je potreba za manjim elementima bila važna zbog bolje prilagodbe mreže terenu. Na slici 3.5. prikazano je umreženje terena.



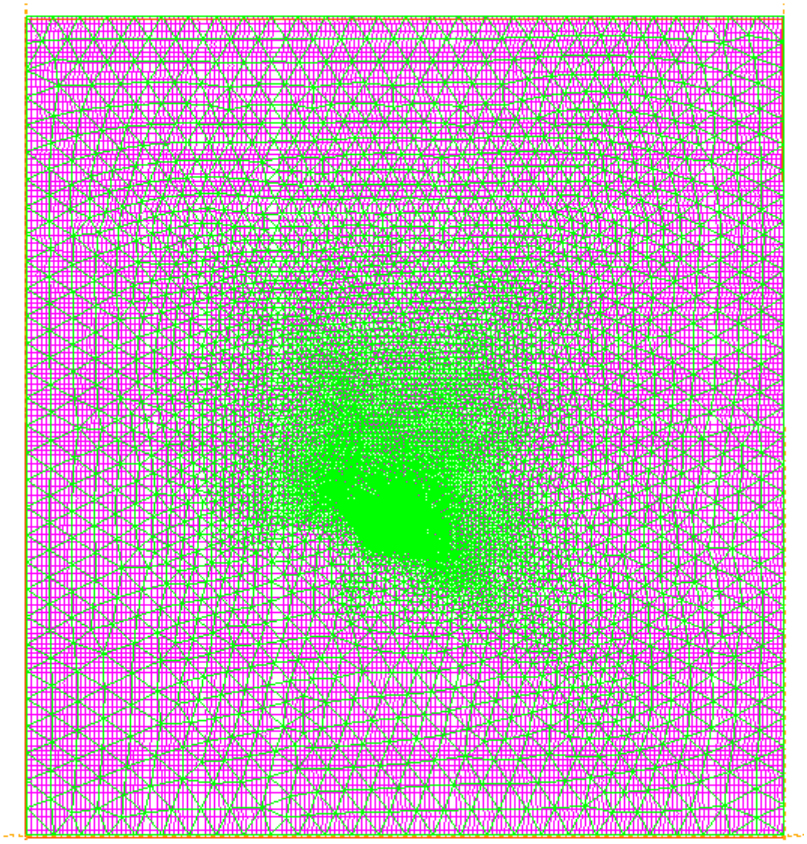
Slika 3.1. Prikaz početnog kutijastog modela generiranog u I-DEAS-u



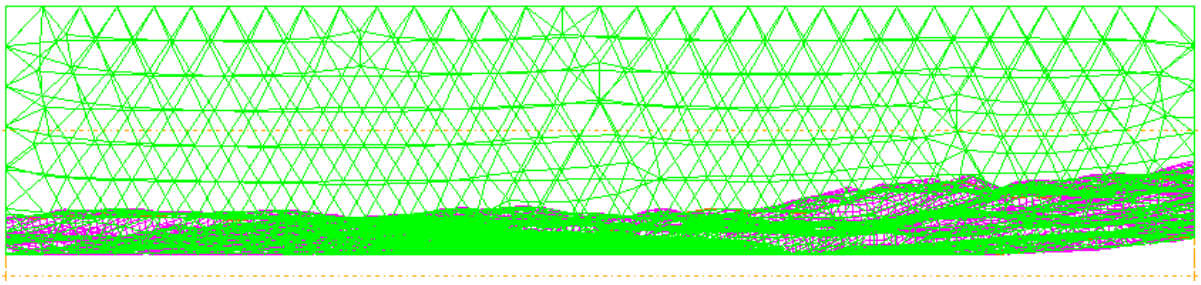
Slika 3.2. Prikaz dimnjaka s istočne strane



Slika 3.3. Prikaz dimnjaka s južne strane

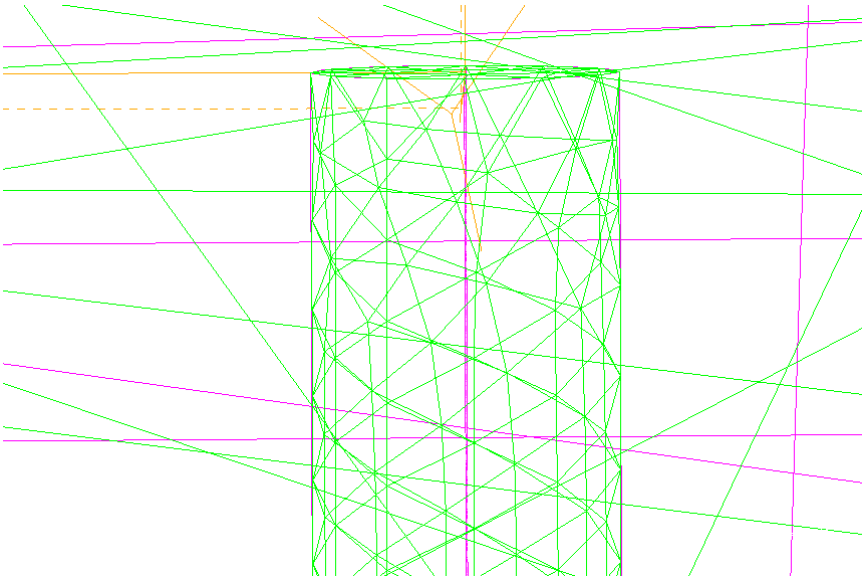


Slika 3.4. Pogled na mrežu sa gornje stranice



Slika 3.5. Pogled na mrežu po rubovima sa južne strane

Moguće je uočiti kako dimnjaci imaju veći broj elemenata pri vrhu, a manji pri dnu te im je izgled blago stožast s vrhom u dnu dimnjaka. To je iz razloga što se s većim brojem elemenata na dnu dimnjaka nije moglo generirati mrežu zbog limitacija koje I-DEAS nameće. Pri korištenju nestrukturirane mreže I-DEAS automatski navlači mrežu tj. smanjuje ili povećava elemente na određenim područjima kako bi bio u stanju dostići broj definiranih čvorova na rubovima. Tako su na dnu dimnjaka FCC definirana dva čvora, a na svim rubnim stranicama po 100 ili više čvorova. U slučaju kada bismo povećali broj čvorova na dnu dimnjaka, a zadržali isti na rubnim stranicama I-DEAS ne bi bio u stanju da izgenerira mrežu jer tražimo od njega preveliki raspon navlačenja koji nije u stanju izvršiti.



Slika 3.6. Prikaz omreženja vrha dimnjaka

4. SIMULIRANJE DINAMIKE PERJANICE SO_2

4.1. NUMERIČKO MODELIRANJE

Analitičke metode kao i inače u drugim područjima znanosti koriste se samo u najjednostavnijim slučajevima fizikalnih ili drugih sistema kako bi opisali određeno stanje toga sustava. Čak i tada se moraju primjeniti određena pojednostavljenja (nazovimo ih pretpostavkama i ograničenjima) kako bi analitička metoda ipak donijela jedno zadovoljavajuće rješenje. Već godinama znanstvenici pokušavaju rješiti jedandžbe koje opisuju fizikalne procese u prirodi i tehnici bez nekog naročitog učinka, ako se radi o općim oblicima jednadžbi kod kojih ne postoje pojednostavljenja. Takve su upravo jednadžbe trodimenzionalnih modela disperzije odnosno sustavi parcijalnih diferencijalnih jednadžbi. Tada nam u pomoć pristižu numeričke metode.

Osnovna razlika numeričkih od analitičkih metoda je da analitikom pokušavamo doći do jedinstvenog rješenja koje će važiti za cijelu domenu, koja se promatra, što znači da se u bilo kojoj točki domene sustava može dobivenim analitičkim rješenjem utvrditi stanje sustava u toj točki. Numeričke metode nasuprot analitičkim razdvajaju cijelu domenu sustava (koja se još u numerici naziva geometrija sustava), koju promatramo, kao konačni broj malih kontrolnih volumena te su u njima i na njihovim rubovima postavljaju zakoni koji u njima vrijede. Temeljna značajka jednog malog kontrolnog volumena je u tome što rješenje u jedinoj njegovoj točki direktno je ovisno o vrijednostima rješenja u točkama kontrolnih volumena koji ga okružuju. Takav pristup, kada od cijele domene dobivamo konačan veliki broj malih kontrolnih volumena, nazivamo diskretizacija sustava. Načini diskretizacije su različiti, od samih odabira oblika kontrolnih volumena te do različitih načina preoblikovanja diferencijalnih jednadžbi u njihove diskretizirane oblike.

Nadalje, potrebno je naglasiti kako diskretizacijom te primjenom diskretiziranih jednadžbi koje opisuju sustav dobivamo rješenja u konačnom broj točaka sustava. Povećanjem broja kontrolnih volumena, tj. smanjenjem njihove veličine, dobivamo veći broj točaka u kojima

možemo dobiti rješenje, što nam govori o tome da se krećemo prema točnijoj slici fizikalnog sustava koji se opisuje. Iz toga se zaključuje da što je veći broj kontrolnih volumena, tim je opis sustava sve točniji. Ograničenje u primjeni velikog broja točaka (u numeričkim metodama točke sustava nazivamo čvorovima) je direktno vezano uz mogućnosti računala koja koristimo. Često je, osim dolaženja do samih rješenja, potrebno to obaviti u što kraćem vremenskom roku. Razlozi tome mogu biti raznoliki: od ograničenog vremena koje nam stoji na raspolaganju za izračunavanje do činjenice da vremenska efikasnost ostavlja nam dovoljno vremena za dorade, konzultacije i poboljšanja.

Numeričke metode i diskretizacija sustava pruža nam mogućnost adaptacije kontrolnih volumena. Naime, kao što smo ranije iznijeli, vrijednost rješenja u nekom čvoru (točki) diskretizirane domene ovisi i mijenja se ovisno o vrijednosti rješenja u susjednim čvorovima. Ako su razlike u vrijednostima susjednih čvorova velike (veliki gradijenti) tada je potreba da, za opisivanje što točnijih vrijednosti varijabli sustava, povećamo broj kontrolnih volumena nad tom domenom (što direktno znači da će ti kontrolni volumeni biti manji od prethodnih) kako bi smanjili velike gradijente na tom području (poboljšavanje – eng. *refinement*). Suprotno velikim gradijentima postoje na određenim dijelovima domene također i gradijenti koji su dovoljno mali te približno jednaki kroz niz od nekoliko čvorova u redu. Kod takvih čvorova s malim gradijentima postoji mogućnost da takvih nekoliko kontrolnih volumena spojimo u jedan volumen (ugrubljivanje – eng. *coarsening*).

Ovaj kratki uvod u numeriku dovoljno je informativan za opis načina modeliranja koji će se koristiti. Danas u svijetu postoji velik broj vrlo sofisticiranih aplikacija za numeričke analize vezane uz specifična područja tehničkih znanosti. Jedan od najsofisticiranijih u području računarske dinamike fluida, a koji će se koristiti, je softverski paket FLUENT.

4.2. FLUENT

FLUENT je računalni program za modeliranje strujanja fluida i prijenosa topline u kompleksnoj geometriji. FLUENT omogućuje potpunu fleksibilnost mreže kojom rješava korisnički problem s nestrukturiranim mrežama koje mogu biti generirane u kompleksnoj geometriji s relativnom lakoćom. Tipovi mreža koje podržava uključuju: 2D trokutaste/kvadrilateralne, 3D tetragonalne/heksagonalne/piramidaste/prizmaste i mješane (hibridne) mreže. FLUENT također omogućuje ugrubljivanje i poboljšavanje mreže na temelju rezultata strujanja. Takva, rješenjem adaptivna, sposobnost mreža naročito je važna za točnije predviđanje strujnog polja u područjima velikih gradijenata kao što su slojevi slobodnog trenja i rubni slojevi. Usporedno rješenju dobivenom strukturiranom mrežom ova sposobnost značajno smanjuje potrebno vrijeme za generiranje dobre mreže.
FLUENT je napisan u C programskom jeziku i omogućuje potpunu fleksibilnost i snagu koju pruža ovaj programski jezik. Prema tome omogućeno je pravo dinamičko alociranje memorije, efektivno strukturiranje podataka i fleksibilna kontrola konvergencije sustava fizikalnih jednadžbi. U dodatku FLUENT koristi klijent/server arhitekturu koja omogućuje rad FLUENT-a kao odvojenih istovremenih procesa na klijentovim sučeljima i snažnim serverima. Radi efikasnog korištenja sadržava interaktivno kontroliranje procesa te potpunu fleksibilnost računala ili operativnog sustava.

Sve funkcije potrebne za dobivanje rješenja i prikaz rezultata pristupne su kroz interaktivno, menu-ima vođeno, sučelje. Korisničko sučelje je napisano u Scheme jeziku (dijalekt LISP-a). Napredni korisnik može korištenjem Scheme jezika prilagoditi i poboljšati sučelje programirajući makro datoteke menu-a i funkcija.

FLUENT rješavač posjeduje mogućnosti modeliranja:

- strujanje u 2D ili 3D geometrijama koristeći nestrukturitanu, rješenjem adaptivnu, mrežu sa svim tipovima elemenata prije spomenutim.
- strujanje stišljivog i nestišnjivog fluida
- stacionarna tranzijentna analiza
- laminarna i turbulentna strujanja
- prijenos topline konvekcijom (uključujući i prirodnu i nametnutu konvekciju)
- njutonovsko i nenjutonovsko strujanje
- konduktivni-konvektivni prijenos topline
- prijenos topline radijacijom
- mješanje i reakcije kemijskih spojeva uključujući podmodele izgaranja i modeliranje reakcija površinskog taloženja
- modeliranje proizvoljnih volumetrijskih izvora masa, momenata, topline, turbulencije i kemijskih spojeva.
- modeliranje strujanja kroz porozan materijal
- dvofazno strujanje (uključujući kavitaciju)
- strujanje slobodne površine s kompleksnim oblicima ploha
- i drugo.

Kako bi riješio sve te modele FLUENT koristi numerički diskretizirane jednadžbe očuvanja mase, količine gibanja, energije, zatim transportne jednadžbe kemijskih spojeva i jednadžbe turbulentnih modela korištene za zatvaranje sustava jednadžbi i dr. Svaka od ovih jednadžbi dolazi u općem formatu koji se sastoji od svih utjecajnih faktora. Utjecajne faktore uključuje korisnik prema tome što zahtjeva fizikalni sistem koji modelira. Također svaka od ovih jedandžbi dolazi i u svom transformiranom stanju (za cilindrični koordinativni sustav te sferni sustav) .

Za naš slučaj uključene su sve jednadžbe te je odabran k-e turbulentni model koji se sastoji od dvije nove jednadžbe.

a) Jednadžba o očuvanju mase

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_i}(\rho u_i) = S_m \tag{1}$$

vrijedi kako za stišljive tako i za nestišljive fluide. S_m je masa dodana kontinuiranoj fazi od druge dispergirane faze (isparivanje kapljica vode).

b) Jednadžba o očuvanju količine gibanja

$$\frac{\partial}{\partial t}(\rho u_i) + \frac{\partial}{\partial x_j}(\rho u_i u_j) = -\frac{\partial p}{\partial x_i} + \frac{\partial \tau_{ij}}{\partial x_j} + \rho g_i + F_i \tag{2}$$

gdje je p statički tlak, g_i gravitacijska sila tijela, F_i vanjska sila tijela (nastaje kao produkt interakcije sa dispergiranom fazom). F_i je također i izvor kao porozni medij ili korisnički definirani izvor. τ_{ij} je tenzor naprezanja dan sa

$$\tau_{ij} = \left[\mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\right] - \frac{2}{3}\mu\frac{\partial u_l}{\partial x_l}\delta_{ij} \tag{3}$$

u čemu je μ molekularna viskoznost, a drugi član razlike utjecaj diletacije volumena.

c) Jednadžba o očuvanju energije

$$\frac{\partial}{\partial t}(\rho E) + \frac{\partial}{\partial x_i}(u_i(\rho E + p)) = \frac{\partial}{\partial x_i}\left(k_{eff}\frac{\partial T}{\partial x_i} - \sum_{j'} h_{j'} J_{j'} + u_j(\tau_{ij})_{eff}\right) + S_h \tag{4}$$

gdje je

k_{eff} – *efektivni konduktivitet* ($k_{eff} = k + k_t$);

k_t – trubulentni toplinski konduktivitet određen turbulentnim modelom koji se koristi;

$J_{j'}$ - difuzijski fluks kemijskog spoja *j'*;

$(\tau_{ij})_{eff}$ - devijatorni tenzor naprezanja

Prva tri izraza s desne strane jednadžbe predstavljaju konduktivitet, difuziju kemijskih spojeva i viskozna disipacija (oslobađanje toplinske energije uslijed viskoznog trenja).

S_h – uključuje toplinu kemijskih reakcija i bilo kojeg drugog volumentrijskog izvora topline definiranog od strane korisnika

U jednadžbi (4)

$$E = h - \frac{p}{\rho} + \frac{u_i^2}{2} \tag{5}$$

gdje je entalpija definirana za idealni plin kao $h = \sum_{j'} m_{j'} h_{j'}$ te za nestišljivi fluid kao $h = \sum_{j'} m_{j'} h_{j'} + \frac{p}{\rho}$ gdje je $m_{j'}$ udjel mase j-tog kem. spoja te $h_{j'} = \int_{T_{ref}}^{T} c_{p,j'} dT$ gdje je T_{ref} = 298.15 K referentna atmosferska temperatura.

d) Transportne jednadžbe za kemijske spojeve (opće jednadžbe difuzije i konvekcije)

Kada modeliramo difuziju i konvekciju onečišćivača potrebno je uključiti konzervativne jedandžbe za kemijske spojeve (onečišćivače). Tada je potrebno definirati

maseni udio kemijskog spoja, $m_{j'}$, unutar jednadžbe konvekcije i difuzije za taj spoj. Takva diferencijalna jednadžba glasi:

$$\frac{\partial}{\partial t}(\rho m_{i'}) + \frac{\partial}{\partial t}(\rho u_i m_{i'}) = -\frac{\partial}{\partial x_i} J_{i',i} + R_{i'} + S_{i'} \qquad (6)$$

gdje je

$R_{i'}$ - količina mase koja nastane ili nestane prilikom kemijskih reakcija

$S_{i'}$ - količina mase nastale dodatkom iz dispergirane faze ili svakog korisnički definiranog izvora mase

Ako u model uključimo N kem. spojeva ili elemenata bit će uključeno N-1 jednadžbi u rješavač.

e) K-e turbulentni model

Za zatvaranje sustava odgovornih jednadžbi odabran je najjednostavniji turbulentni model. To je standardni k-e turbulentni model kojeg predstavljaju dvije transportne jednadžbe u kojima se potpuno odvojeno riješavaju dvije nove varijable: brzina turbulencije izražena kinetičkom energijom turbulencije k i turbulentna dužina izražena količinom disipacije turbulencije ε. Ovaj model je jedan od najpraktičnijih modela za proračune strujanja fluida. Njegova snaga, ekonomičnost te zadovoljavajuća točnost nad širokim područjem turbulentnih strujanja objašnjava njegovu popularnost u industrijskim strujanjima i simulacijama prijenosa topline. To je poluempirički model čije derivacije jednadžba modela ovise o fenomenološkim razmatranjima i empirizmu.
Derivacija transportnih jednadžbi za k i ε provedena je pretpostavkom da je cijelo područje domene modela potpuno turbulentno i efekti molekularne viskoznosti zanemarivi te je stoga ovaj model primjenjiv samo za potpuno turbulentna strujanja.

Kinetička turbulentna jednadžba je

$$\rho \frac{Dk}{Dt} = \frac{\partial}{\partial x_i}\left[\left(\mu + \frac{\mu_t}{\sigma_k}\right)\frac{\partial k}{\partial x_i}\right] + G_k + G_b - \rho\varepsilon - Y_M \qquad (7)$$

gdje je

G_k - produkcija kinetičke energije zbog gradijenata osrednjene brzine definirana kao

$G_k = -\rho\overline{u_i' u_j'}\frac{\partial u_j}{\partial x_i}$;

G_b - produkcija kinetičke energije turbulencije zbog uzgona definirana kao

$G_b = \beta g_i \frac{\mu_t}{\Pr_t}\frac{\partial T}{\partial x_i}$;

β - koeficijent toplinske ekspanzije;

μ_t - turbulentna viskoznost (eng. "*eddy viscosity*") $\mu_t = \rho C_\mu \frac{k^2}{\varepsilon}$ gdje je $C_\mu = 0.09$

$\Pr_t$ - turbulentni Prandtlov broj za energiju koji iznosi 0.85;

Y_M - doprinos diletacije fluktuacije u turbulentnom stišljivom fluidu na globalnu količinu disipacije (taj je doprinos vezan za stišljivi fluid pri visokim Machovim brojevima);

σ_k - turbulentni Prandtlov broj za kinetičku energiju turbulencije ($\sigma_k = 1$).

Turbulentna energija disipacije proračunava se prema

$$\rho \frac{D\varepsilon}{Dt} = \frac{\partial}{\partial x_i}\left[\left(\mu + \frac{\mu_t}{\sigma_\varepsilon}\right)\frac{\partial \varepsilon}{\partial x_i}\right] + C_{1\varepsilon}\frac{\varepsilon}{k}\left(G_k + C_{3\varepsilon}G_b\right) - C_{2\varepsilon}\rho\frac{\varepsilon^2}{k} \qquad (8)$$

gdje su

$C_{1\varepsilon}, C_{2\varepsilon}, C_{3\varepsilon}$ - konstante te iznose $C_{1\varepsilon} = 1.44$, $C_{2\varepsilon} = 1.92$ te $C_{3\varepsilon}$ definiran kao stupanj utjecaja uzgona na disipaciju $C_{3\varepsilon} = \tanh\left|\frac{v}{u}\right|$ gdje je v projekcija brzine strujanja na vertor gravitacijskog ubrzanja;

σ_ε - turbulentni Prandtlov broj za količinu disipacije turbulencije ($\sigma_\varepsilon = 1.3$).

Također odabirom standardnog k-e modela u jednadžbu energije (4) definiran je turbulentni toplinski konduktivitet $k_t = \frac{c_p \mu_t}{\mathrm{Pr}_t}$.

4.3. POSTUPAK MODELIRANJA U FLUENT-U ZA NESTACIONARNE METEOROLOŠKE UVJETE

4.3.1. Rubni uvjet dobiven iz korisnički definirane funkcije

Modeliranje u FLUENT-u zahtjeva niz pripremnih radnji i podešavanje brojnih parametara prije nego se pokrene rješavač sustava jednadžbi u diskretiziranoj domeni modela. Ti koraci su sljedeći:

1. Kreiranje geometrije modela;

2. Pokretanje 2D ili 3D rješavača u FLUENT-u;

   Kada pokrećemo FLUENT, potrebno je naznačiti kako želimo modelirati. Dvodimenzionalni rješavač se odnosi na modeliranje u jednoj ravnini tj. u dvodimenzionalnom prostoru. Primjeri takvih modela su razna strujanja u cilindričnim cijevima te u strujnim poljima gdje na varijable koje promatramo ne utječe ili malo utječe efekt treće dimenzije. Zbog toga iz jednadžbi ispadaju svi efekti vezani uz treću *z* dimenziju. Za razliku od 2D postoji odabir 3D modeliranja kod kojeg postoje promjene varijabli u sve tri dimenzije. Za ovaj slučaj odabran je 3D rješavač.

3. Unos geometrije;

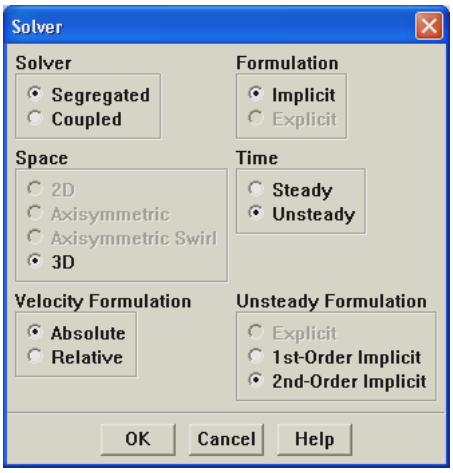
   Geometrija se učitava korištenjem naredbe (*File->Import->I-DEAS Universal file*) jer smo iz I-DEAS-a iznijeli geometriju upravo u tom formatu.

4. Provjera i izmjena geometrije;

Prilikom transfera geometrija se može izmjeniti te je potrebno odmah nakon unosa geometriju provjeriti: uočiti da su svi izračunati volumeni geometrije pozitivnog predznaka jer bi u slučaju negativnih predznaka rješavač iznio krive tj. neželjene rezultate (*Grid->Check*). Također potrebno je iznova definirati ishodišnje koordinate te voditi računa o eventualnoj izmjeni koordinatnih osi (*Grid->Translate*). Nadalje geometrija koju smo učitali sastoji se od jednog bloka te je bilo potrebno odvojiti pojedine stranice domene kako bi ih kasnije identificirali i dodijelili im odgovarajući rubne uvjete (*Grid->Separate->Faces, Grid->Merge*). Tijekom cijele provjere i izmjene geometrije vizualno smo pratili izgled geometrije kako bi zaista bili uvjereni da sve ide kao što smo i zamislili na početku (*Display->Grid*).

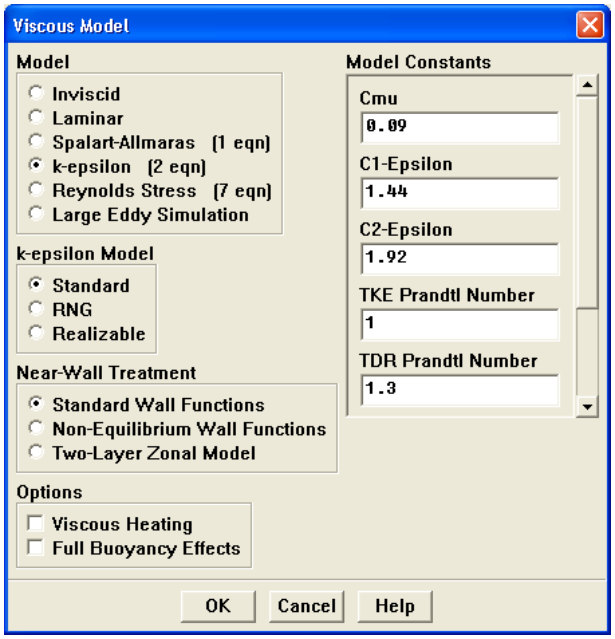
5. Odabir definicije rješavača;

Odabran je individualni rješavač (*Define->Models->Solver->Segregated*) koji je standardan za FLUENT i kod kojeg se sve zakonite jednadžbe rješavaju zasebno jedna od druge. Postoji i mogućnost da se pojedine jednadžbe rješavaju simultano (rješavaju se istovremeno jednadžbe količine gibanja i energije, eng. *Implicit coupled solver*). Budući da se rubni uvjeti mijenjaju u ovisnosti o vremenu, u meniju *time* odabrali smo opciju *unsteady* (slika 4.3.1.).



Slika 4.3.1. Odabir vrste rješavača

6. Odabir osnovnih jednadžbi;



Slika 4.3.2. Odabir viskoznog modela

Odabran je turbulentni *k-e* model (*Define->Models->Viscous->k-epsilon->Standard*), postojanje kemijskih spojeva (što uključuje rješavanje transportnih jednadžbi konvekcije i difuzije pojedinih kemijskih spojeva iz poglavlja 4.2.) (*Define->Models->Species->Multiple Species*) koja automatski uključuje

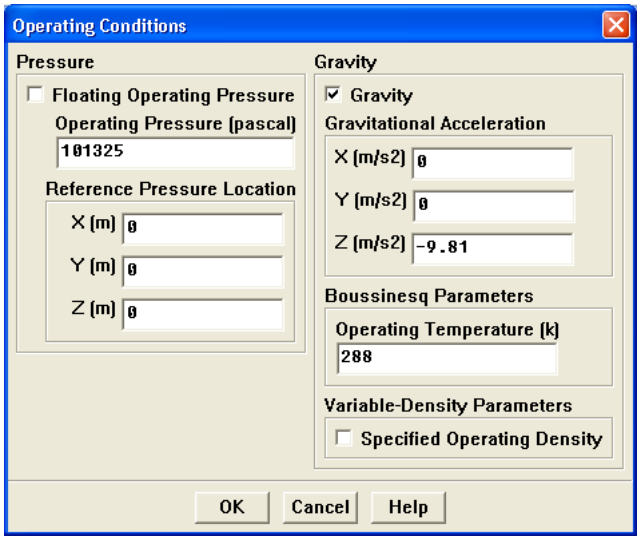
energetsku jednadžbu zbog postojanja razlika u gustoćama i temperaturi pojedinih kemijskih spojeva koje ćemo uključiti.

Postoji još niz mogućnosti: od odabira laminarnog strujanja te niza drugih turbulentnih modela, radijacije, diskretne faze te određenih polutanata kao NOx i čađe do korisnički definiranih veličina kao poroznih materijala, izmjenjivača topline i dr.

7. Definicija materijala i njihovih osobina;

Iz baze podataka odabrana je za tip materijala mješavina u koju su iz baze podataka o materijalima uključeni sumporni dioksid i zrak. (*Define->Materials*)

8. Definicija operativnih uvjeta;



Slika 4.3.3. Odabir operativnih uvjeta

Definiran je referentni tlak od 101325 Pa, te gravitacijska akceleracija $g_x = g_y = 0$; $g_z = -9.81\ ms^{-1}$.

9. Unos profila brzina;

Kako bi definirali promjenu brzine u u određenom vremenskom periodu trebalo je napisati sljdeći *kôd* u programskom jeziku C .

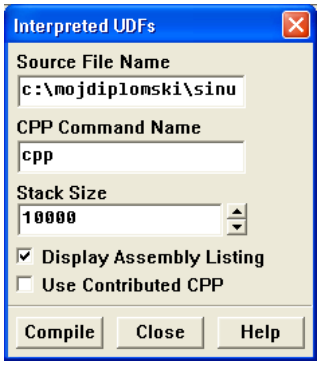
```
include "udf.h"

DEFINE_PROFILE(inlet_xv_sinusoidal, /* function name   */

            thread,          /* thread          */
            nv)              /* variable number */
{
 face_t f;
 real flow_time = RP_Get_Real("flow-time");

 begin_f_loop (f,thread)
   {
     F_PROFILE(f,thread,nv) = 7. + 2.1*sin(0.034906585*flow_time);
   }
 end_f_loop (f,thread)
}
```

U ovom *kôdu* promjena brzine vjetra definirana je kao sinusna funkcija

*F_PROFILE(f,thread,nv) = 7. + 2.1*sin(0.034906585*flow_time);*

gdje je srednja vrijednost brzine 7 m/s, amplituda je 30% srednje vrijednosti tj. 2.1 m/s, izraz u zagradi funkcije *sin* predstavlja vremenski korak od 180 s.

Zatim, napisani *kôd* treba ubaciti u FLUENT. To se čini odabirom sljedećih menija: (*Define->User-Defined->Functions->Interpreted*).



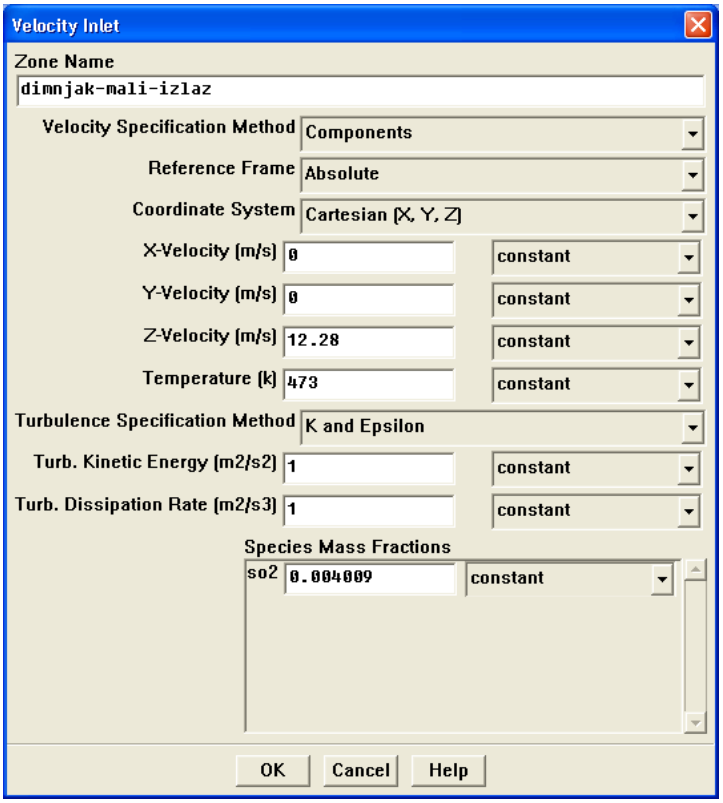
Slika 4.3.4. Unos korisničke funkcije

Dobivamo prozor gdje treba navesti ime funkcije koju želimo ubaciti i mjesto gdje se nalazi. Odabirom opcije *compile*, učitali smo korisničku funkciju u FLUENT.

10. Definicija rubnih uvjeta;

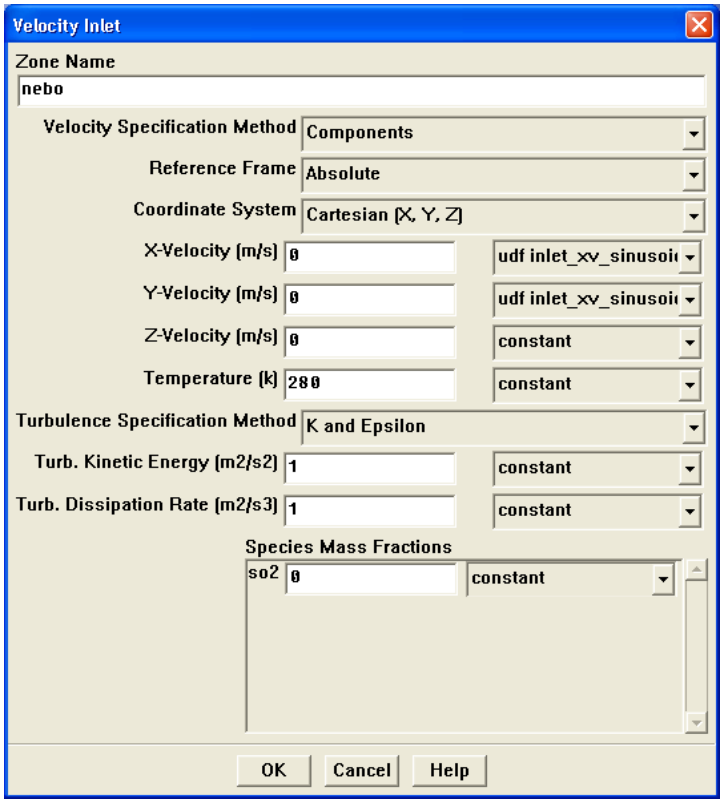
Definicija rubnih uvjeta ovisi o našem predviđanju kako će strujanje koje modeliramo izgledati, tj. što predstavlja ulazne presjeke strujnog polja, a što izlazne presjeke, što je zid (nema protoka kroz zidove) te da li postoje ravnine simetrije ili ne. Ulazni i izlazni presjeci mogu biti definirani na razne načine u FLUENT-u.

Kao ulazne presjeke (rubni uvjet: ulaz brzine) (eng. *Velocity inlet*) definirali smo vrhove dimnjaka kao izlazne presjeke onečišćivača iz dimnjaka te je definirana izlazna brzina, temperatura i maseni udio sumpornog dioksida.



Slika 4.3.5. Odabir rubnih uvjeta na izlazu iz dimnjaka

Teren i plaštevi obiju dimnjaka su definirani kao zid (eng. *Wall*) s definiranim stupnjem površinske hrapavosti 0.5. Gornja stranica geometrije modela definirana je učitanim profilom vjetra kao rubni uvjet: *ulaz brzine*.



Slika 4.3.6. Odabir rubnih uvjeta na gornoj stranici modela

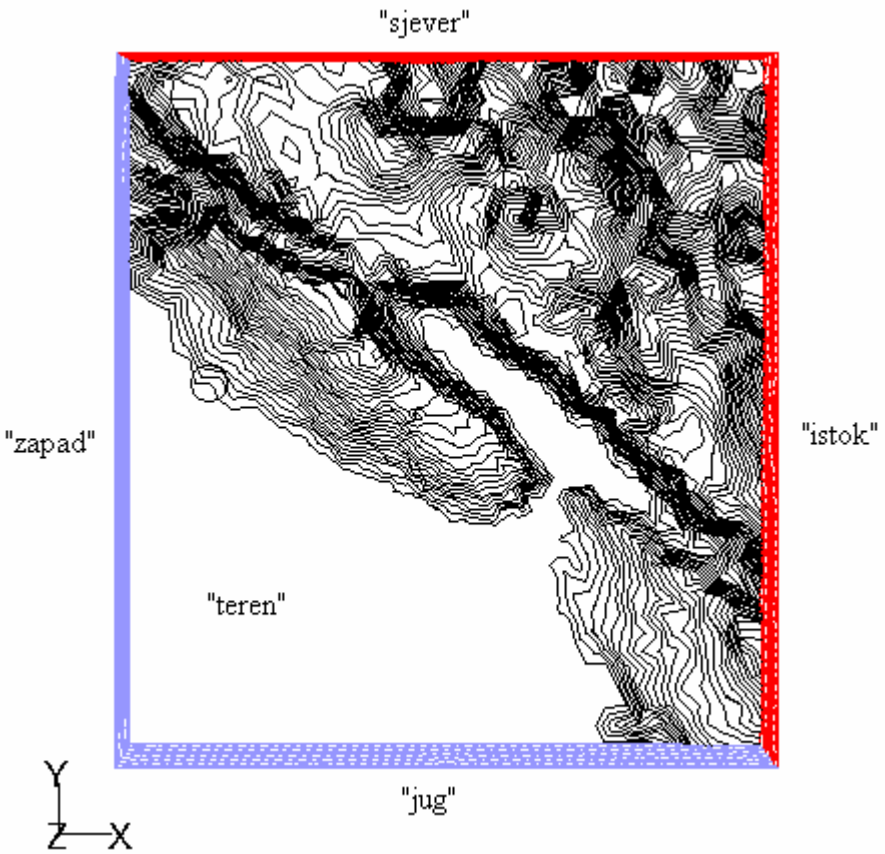
Bočne stranice su definirane različito, ovisno o smjeru vjetra. Bočne stranice od kojih puše vjetar definirane su profilima brzine i smjerom vektora brzine kao rubni uvjeti: ulaz brzine. To su u našem slučaju stranice "zapad" i "jug" (Slika 4.3.7.). Bočne stranice prema kojima puše vjetar te dispergira onečišćivače poprimile su rubni uvjet: strujni izlaz (eng. *Outflow*). To su stranice "sjever" i "istok".

Ovakav rubni uvjet korišten je iz razloga što su odgovorne varijable nepoznate na tim stranicama, tj. izlazima iz strujnog polja te ih tako definirani rubni uvjet

predviđa kao jednake susjednim čvorovima unutar domene. Koristeći *strujni izlaz* kao rubni uvjet povećavamo vrijeme konvergencije (prosjek je 120 iteracija) ali

omogućujemo točniju sliku strujnog polja. Ako bi koristili rubni uvjet *tlačni izlaz* (eng. *Pressure outlet*) vrijeme konvergencije bilo bi 50 iteracija, ali imali bi ograničene izlazne presjeke iz strujnog polja nametnutim tlakom i temperaturom.

Također je obavljeno spajanje stranica "sjever" i "istok" koje su poprimile rubni uvjet strujni izlaz, iz razloga što je za *strujni izlaz* potrebno definirati količinu izlaznog materijala kroz pojedinu stranicu. Kako je ta količina za pojedinu stranicu nepoznata spojene su sve u jednu te je na njoj definirana količina izlaza jednaka jedinici tj. potpunom izlazu materijala kroz nju.



Slika 4.3.7. Prikaz rubnih stranica modela

11. Odabir kontrolnih parametara rješavača

Kontrolni parametri rješavača omogućuju uključivanje ili isključivanje pojedinih jednadžbi iz proračuna, definiciju faktora podrelaksacije rješavača sustava diferencijalnih jednadžbi te odabir načina diskretizacije sustava jednadžbi. U našem slučaju nismo mijenjali definirane parametre. Tijekom proračuna modificirani su neki od podrelaksirajućih faktora kako bi se ubrzala konvergencija jednadžbi (*Solve->Control->Solution*).

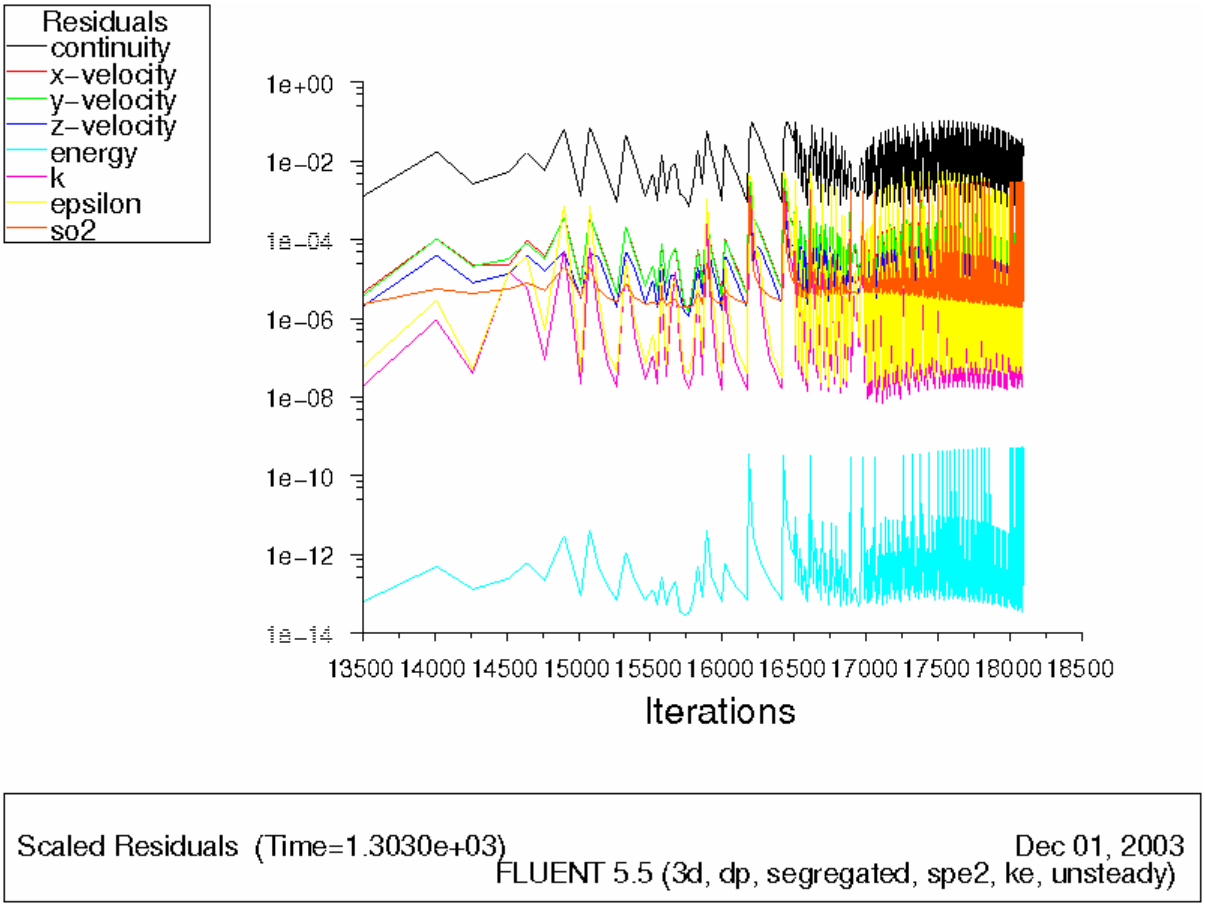
12. Inicijalizacija

Inicijalizacijom se definiraju početni uvjeti za rješavač sistema diferencijalnih jednadžbi. Nad cijelom domenom (u svakom čvoru) definiraju se tada početne vrijednosti varijabla koje se rješavaju, a moguće je definirati početne uvjete korisničkim unošenjem vrijednosti varijabli ili odabirom vrijednosti varijabli nad jednom stranicom modela čije smo rubne uvjete netom prije podesili. Za sve smjerove vjetra odabrali smo vrijednosti rubnih uvjeta stranice koja predstavlja ulaz vjetra u strujno polje.

13. Pokretanje rješavača

Otvaranjem menija (*Solve->Iterate*) dobivamo prozor u kojem trebamo definirati veličinu vremenskog koraka (eng. *time step size*) i broj vremenskih koraka (eng. *number of time steps*). Vremeski korak mijenjamo da bi smanjili broj iteracija po vremenskom koraku. Idealan broj iteracija po vremenskom koraku je 15-20. Ako FLUENT treba više iteracija po vremenskom koraku da bi konvergirao, vremenski korak je prevelik. Rješavač na grafičkom sučelju FLUENT-a prikazuje razliku vrijednosti trenutne i prethodne iteracije svake varijable. U numeričkim metodama kako iteracije teku tako te razlike, tj. “greške” opadaju dok sve ne postignu limite koji su im definirani (*Solve->Controls-*

>*Limits*). Za jednadžbu kontinuiteta, jednadžbe količine gibanja, transportne jednadžbe pojedinih kem. spojeva i turbulentnih modela postavljena je greška zadovoljenja od 0.001, a za jednadžbu očuvanja energije 0.000001. Težnju grešaka ka sve manjim vrijednostima nazivamo konvergencija. Konvergencije pojedinih jednadžbi, osim tekstualnim zapisom, možemo pratiti grafičkim prikazom ovisnosti konvergencije i broja iteracija tzv. Ostataka (eng. *Residuals*). (*Solve->Monitors->Residuals->Plot*) kao prikazano na slici 4.3.8.



Slika 4.3.8. Grafički prikaz ovisnosti konvergencije i broja iteracija

Nakon uspješne konvergencije svih jednadžbi tj. postizanja svih limita rješavač je riješio sustav jednadžbi.

14. Adaptacija

Na početku je bolje krenuti u modeliranje sa nešto grubljom mrežom kako bi konvergencija trajala kraće. Kasnije tijekom analize moguće je ugušćivati mrežu, tj. vršiti adaptaciju za dobivanje točnijih rješenja na željenim područjima domene (manji gradijenti brzina, bolji prikaz difuzije onečišćivača, itd.). Adaptacija koju smo proveli bazirala se na gradijentima brzine nad područje značajnih koncentracija onečiščivača. Takvom adaptacijom bolje je razlučeno područje difuzije onečišćenja. Također je izvršena adaptacija nad vrhom dimnjaka (površinom izlaznog presjeka onečišćivača iz dimnjaka) kako bi se dobilo na većem broju čvorova nad područjem iz kojeg izviru kemijski spojevi. Takva adaptacija po rubovima (eng. *Boundary adaption*) primjenjena je tri puta za tri ćelije nad dimnjakom, potom za dvije i na kraju za jednu.

Nakon izvršene adaptacije mreže prema rješenjem dobivenih rezultata ponovimo postupak pokretanja rješavača bez da smo prije inicijalizirali domenu. Time nastavljamo iz trenutka gdje je prethodna iteracija završila (skokovi u ostacima (*residuals*) Slika 4.3.8.).

4.3.2. Rubni uvjet dobiven iz meteorološkog programa MM5

Drugi način modeliranja dinamike perjanice SO_2 u FLUENT-u je bio da za rubne uvjete koristimo podatke iz meteorološkog programa MM5. Ideja je bila da se za svaki vremenski korak u FLUENT importiraju novi rubni uvjeti dobiveni iz MM5. Dobiveni meteorološki podaci su u obliku tekstualnog file-a za svaka 3 sata. Struktura dobivenih podataka izgleda u tekstualnom formatu ovako:

```
+2003-04-10_00:00:00.0000        0.00000 Hours
        U      3  23   1   1   1 D    YXP  :        -0.39816628 m/s
        .
        .
        V      3  23   1   1    1 D   YXP  :         5.04290676 m/s
        .
        .
        T      3  23   1   1   1 C    YXP  :       277.83633423 K
        .
        .
        W      3  23   1   1   1 C    YXP  :         0.01953787 m/s
        .
        .
        PP     3  23   1   1   1 C     YXP  :      1258.25109863 Pa
        .
        .
        H      3  23  18  20   1 C    YXP  :     16055.72265625 m
```

gdje U, V, W predstavljaju komponente brzine; T predstavlja temperaturu; PP predstavlja tlak; H predstavlja visinu u atmosferi za koju se navedeni podaci odnose. Dobiveni podaci bili su za puno veće područje terena nego što je naš model. Na osnovi veličine modela terena (dužina, širina, visina) bilo je potrebno filtrirati podatke iz meteorološkog programa i odabrati samo one čije apsolutne koordinate ulaze u područje našeg modela terena. Sljedeći problem koji je trebalo riješiti je transformacija podataka iz gore prikazanog primjera tekstualnog file-a u format koji koji se može importirati u FLUENT. Također je bilo potrebno napraviti linearnu interpolaciju meteoroloških podataka jer je vremenski razmak između njih bio prevelik (svaka 3 sata). Vremenski razmak između meteoroloških podataka je bitan radi konvergencije rješavača u FLUENT-u (što je manji

vremenski korak, to će konvergencija biti brža). Budući da nakon linearne interpolacije dobivamo jako veliki broj file-ova sa meteorološkim podacima, bilo bi nepraktično svaki file pojedinačno ubacivati za svaki vremeski korak. To se može riješiti korištenjem journal file-a. Učitavanjem journal file-a u FLUENT moguće je automatizirati postupak unošenja profila sa podacima. Sve što je potrebno, je upisivanje odgovarajućih naredbi u journal file i upisivanje profila koje želimo da journal file automatski učitava. Primjer journal file-a za naš slučaj izgleda ovako:

```
fi rp profili\profili0.prof
so dti 1 100
fi rp profili\profili1.prof
so dti 1 100
fi rp profili\profili2.prof
so dti 1 100
.
.
.
```

Transformacija podataka u tekstualni format koji se može importirati u FLUENT, linearna interpolacija meteoroloških podataka, automatsko generiranje profila sa meteorološkim podacima dobivena je sljedećom *kôdom* napisanim u u programskom jeziku C++:

```
include <iostream>
#include <iomanip>
#include <fstream>
#include <string>
#include <map>
#include <vector>

using namespace std;

const int xMin=15;
const int xMax=23;

const int zMin=1;
const int zMax=6;

const int yMin=1;
const int yMax=11;
```

```
const int numN=180; // broj između 90 = 2 min  180 = 1 min

const double xMinValue=0;
const double xMaxValue=9644;


const double yMinValue=0;
const double yMaxValue=10431;

const char *outputName="profili";
const char *jouFileName="profili.jou";

struct data
{
	double vx;
	double vy;
	double vz;
	double temperature;
	double pressure;
	double z;
};

struct ProfileInfo
{
	string fileName;
	int xMin;
	int xMax;
	int yMin;
	int yMax;
	int zMin;
	int zMax;
	int inletCondition;
	int lastInletStatus;
};

typedef map<string,data> DataCont; // string xcoordinata ykoordinata index npr 3 2

typedef map<string,DataCont> ProfileCont;

typedef map<int,ProfileCont> TimeProfileCont;

typedef map<string,ProfileInfo> ProfileInfoCont;

TimeProfileCont p;
ProfileInfoCont pinfo;

string koordString(int x, int y, int z)
{
	char buffer[256];
	sprintf(buffer,"%d %d %d",x,y,z);
	return string(buffer);
}

void readProfileData(const string &profileName)
```

```
{
        string line;

        int time=-1;

        ifstream input(pinfo[profileName].fileName.c_str());

        if(!input.is_open())
        {
                cout<<"Cannot open"<<pinfo[profileName].fileName<<endl;
                return;
        }


        int xmin=pinfo[profileName].xMin;
        int xmax=pinfo[profileName].xMax;
        int ymin=pinfo[profileName].yMin;
        int ymax=pinfo[profileName].yMax;
        int zmin=pinfo[profileName].zMin;
        int zmax=pinfo[profileName].zMax;

        while(!input.eof())
        {
                getline(input,line);

                if(line.length()>0)
                {
                        if(line[0]=='+')
                        {
                                time++;
                        }
                        else
                        {
                                int x,y,z;
                                double value;
                                char dbuff[256];
                                int dint;
                                char rtype[256];

        sscanf(line.c_str(),"%s%d%d%d%d%d%s%s%s%lf",rtype,&dint,&x,&y,&z,&dint,dbuff,dbuff,dbuff,&
value);

                                if(x>=xmin && x<=xmax && y>=ymin && y<=ymax && z>=zmin &&
z<=zmax)
                                {
                                        switch (rtype[0])
                                        {
                                        case 'U':
                                                p[time][profileName][koordString(x,y,z)].vy=value;
                                                break;
                                        case 'V':
                                                p[time][profileName][koordString(x,y,z)].vx=value;
                                                break;
                                        case 'W':
                                                p[time][profileName][koordString(x,y,z)].vz=value;
```

```
                                        break;
                                    case 'P':

            p[time][profileName][koordString(x,y,z)].pressure=value;
                                        break;
                                    case 'T':

            p[time][profileName][koordString(x,y,z)].temperature=value;
                                        break;
                                    case 'H':
                                        p[time][profileName][koordString(x,y,z)].z=value;
                                        break;
                                    default:
                                        cout<<"Error in line "<<line<<endl;
                                        getchar();
                                    }
                                }
                            }
                        }
            }
}

double xValue(int i)
{
            return xMinValue+(i-xMin)*(xMaxValue-xMinValue)/(xMax-xMin);
}

double yValue(int j)
{
            return yMinValue+(j-yMin)*(yMaxValue-yMinValue)/(yMax-yMin);
}

void writeProfiles()
{
            int counter=0;

            ProfileInfoCont::iterator iter;

            int maxtime=p.size()-1;

            int t,ti;

            ofstream inletlog("inlet.log");
            ofstream promjenelog("promjene.log");
            ofstream joufile(jouFileName);

            for(t=0;t<maxtime;t++)
            {

                        vector<double> x;
                        vector<double> y;
                        vector<double> z;
                        vector<double> tt;
                        vector<double> pp;
                        vector<double> vx;
```

```
vector<double> vy;
vector<double> vz;

vector<double> x1;
vector<double> y1;
vector<double> z1;
vector<double> tt1;
vector<double> pp1;
vector<double> vx1;
vector<double> vy1;
vector<double> vz1;

vector<double> x2;
vector<double> y2;
vector<double> z2;
vector<double> tt2;
vector<double> pp2;
vector<double> vx2;
vector<double> vy2;
vector<double> vz2;

int count;

if(t<(maxtime-1))
{
	count=numN;
}
else
{
	count=numN+1;
}



for(ti=0;ti<count;ti++,counter++)
{
	double f=(double)ti/numN;
	char fileName[256];

	sprintf(fileName,"profili\\%s%d.prof",outputName,counter);
	ofstream out(fileName);

	inletlog<<endl<<"Time "<<counter<<" = "<<counter*60<<endl;


	joufile<<"fi rp "<<fileName<<endl;
	joufile<<"so dti 1 100"<<endl;



	cout<<fileName<<endl;

	for(iter=pinfo.begin();iter!=pinfo.end();iter++)
	{
		string profileName=(*iter).first;
```

```
int n=p[t][profileName].size();


DataCont::iterator dataIter;

x.resize(n);
y.resize(n);
z.resize(n);
tt.resize(n);
pp.resize(n);
vx.resize(n);
vy.resize(n);
vz.resize(n);

x1.resize(n);
y1.resize(n);
z1.resize(n);
tt1.resize(n);
pp1.resize(n);
vx1.resize(n);
vy1.resize(n);
vz1.resize(n);
x2.resize(n);
y2.resize(n);
z2.resize(n);
tt2.resize(n);
pp2.resize(n);
vx2.resize(n);
vy2.resize(n);
vz2.resize(n);
int i;

for(i=0,dataIter=p[t][profileName].begin();
dataIter!=p[t][profileName].end(); dataIter++, i++)
{
        int xi,yi,zi;
        sscanf((*dataIter).first.c_str(),"%d%d%d",&xi,&yi,&zi);

        data &d=(*dataIter).second;

        x1[i]=xValue(xi);
        y1[i]=yValue(yi);
        z1[i]=d.z;
        tt1[i]=d.temperature;
        pp1[i]=d.pressure;
        vx1[i]=d.vx;
        vy1[i]=d.vy;
        vz1[i]=d.vz;
}

for(i=0,dataIter=p[t+1][profileName].begin();
dataIter!=p[t+1][profileName].end(); dataIter++, i++)
{
```

```
        int xi,yi,zi;
        sscanf((*dataIter).first.c_str(),"%d%d%d",&xi,&yi,&zi);

        data &d=(*dataIter).second;

        x2[i]=xValue(xi);
        y2[i]=yValue(yi);
        z2[i]=d.z;
        tt2[i]=d.temperature;
        pp2[i]=d.pressure;
        vx2[i]=d.vx;
        vy2[i]=d.vy;
        vz2[i]=d.vz;
}


double sum_vx=0.0;
double sum_vy=0.0;
double sum_vz=0.0;
for(i=0;i<n;i++)
{
        x[i]=x1[i];
        y[i]=y1[i];
        z[i]=z1[i];
        tt[i]=tt1[i]+f*(tt2[i]-tt1[i]);
        pp[i]=pp1[i]+f*(pp2[i]-pp1[i]);
        vx[i]=vx1[i]+f*(vx2[i]-vx1[i]);
        vy[i]=vy1[i]+f*(vy2[i]-vy1[i]);
        vz[i]=vz1[i]+f*(vz2[i]-vz1[i]);

        sum_vx+=vx[i];
        sum_vy+=vy[i];
        sum_vz+=vy[i];
}



int inlet=0;


switch(pinfo[profileName].inletCondition)
{
case 0:
        if(sum_vx>0.0) inlet=1;
        break;
case 1:
        if(sum_vx<0.0) inlet=1;
        break;
case 2:
        if(sum_vy>0.0) inlet=1;
        break;
case 3:
        if(sum_vy<0.0) inlet=1;
        break;
case 4:
```

```
                        if(sum_vz<0.0) inlet=1;
                        break;
                }

                if(pinfo[profileName].lastInletStatus!=inlet)
                {
                        promjenelog<<endl<<"Time        "<<counter<<"        =
"<<counter*60<<endl;

                        if(inlet)
                        {
                                promjenelog<<profileName<<"  ->  inlet  sum
vx"<<sum_vx<<" sum vy"<<sum_vy<<"sum vz"<<sum_vz<<endl;
                        }
                        else
                        {
                                promjenelog<<profileName<<"  ->  outlet  sum
vx"<<sum_vx<<" sum vy"<<sum_vy<<"sum vz"<<sum_vz<<endl;
                        }

                        pinfo[profileName].lastInletStatus=inlet;
                }

                if(inlet)
                {
                        inletlog<<profileName<<" -> inlet  sum vx"<<sum_vx<<" sum
vy"<<sum_vy<<"sum vz"<<sum_vz<<endl;
                }
                else
                {
                        inletlog<<profileName<<" -> outlet  sum vx"<<sum_vx<<" sum
vy"<<sum_vy<<"sum vz"<<sum_vz<<endl;
                }

                {  // begin writing profiles



                        // oznaka profila i broj tocaka
                        out<<"(("<<profileName<<"                        point
"<<p[t][profileName].size()<<")"<<endl;

                        // x - koordinata
                        out<<"(x"<<endl;

                        for(i=0;i<n;i++)
                        {
                                out<<x[i]<<endl;
                        }

                        out<<")"<<endl;

                        // y - koordinata
                        out<<"(y"<<endl;
```

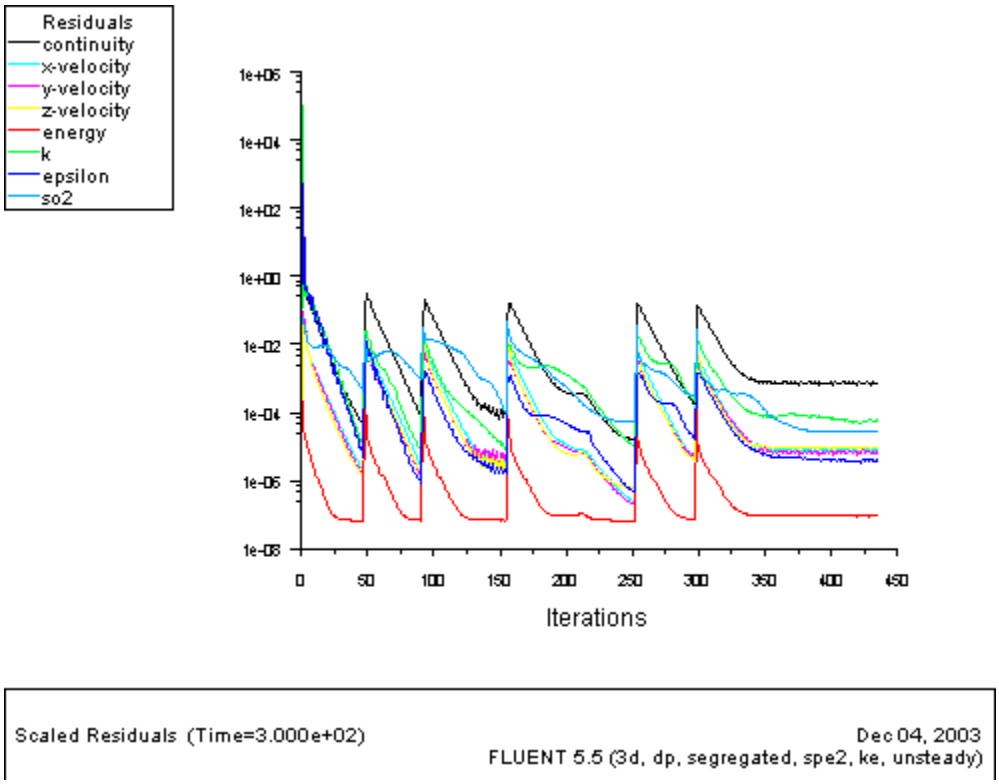
```
for(i=0;i<n;i++)
{
        out<<y[i]<<endl;
}

out<<")"<<endl;

// z - koordinata
out<<"(z"<<endl;

for(i=0;i<n;i++)
{
        out<<z[i]<<endl;
}

out<<")"<<endl;

// total-pressure
out<<"(total-pressure"<<endl;

for(i=0;i<n;i++)
{
        out<<pp[i]<<endl;
}

out<<")"<<endl;

// x-velocity
out<<"(x-velocity"<<endl;

for(i=0;i<n;i++)
{
        out<<vx[i]<<endl;
}

out<<")"<<endl;

// y-velocity
out<<"(y-velocity"<<endl;

for(i=0;i<n;i++)
{
        out<<vy[i]<<endl;
}

out<<")"<<endl;

// z-velocity
out<<"(z-velocity"<<endl;

for(i=0;i<n;i++)
{
        out<<vz[i]<<endl;
}
```

```
                                        out<<")"<<endl;

                                        // total-temperature
                                        out<<"(total-temperature"<<endl;

                                        for(i=0;i<n;i++)
                                        {
                                                out<<tt[i]<<endl;
                                        }

                                        out<<")"<<endl;
                                        out<<")"<<endl;

                                } // end writing profile

                        }
                }
        }

}

void ReadProfilesData()
{
        ProfileInfoCont::iterator iter;

        string profileName;

        for(iter=pinfo.begin();iter!=pinfo.end();iter++)
        {
                profileName=(*iter).first;
                readProfileData(profileName);
        }
}

void AddProfile(const string &name, const string &fileName, int inletCondition, int xMin, int xMax, int yMin,
int yMax, int zMin, int zMax)
{
        pinfo[name].fileName=fileName;
        pinfo[name].xMin=xMin;
        pinfo[name].xMax=xMax;
        pinfo[name].yMin=yMin;
        pinfo[name].yMax=yMax;
        pinfo[name].zMin=zMin;
        pinfo[name].zMax=zMax;
        pinfo[name].inletCondition=inletCondition;
        pinfo[name].lastInletStatus=-1;
}

int main()
{
        AddProfile("istok","istok.txt",(int)1,xMax,xMax,yMin,yMax,zMin,zMax);
        AddProfile("jug","jug.txt",(int)2,xMin,xMax,yMin,yMin,zMin,zMax);
        AddProfile("zapad","zapad.txt",(int)0,xMin,xMin,yMin,yMax,zMin,zMax);
        AddProfile("sjever","sjever.txt",(int)3,xMin,xMax,yMax,yMax,zMin,zMax);
        AddProfile("nebo","nebo.txt",(int)4,xMin,xMax,yMin,yMax,zMax,zMax);
```

```
        ReadProfilesData();
        writeProfiles();
        return 0;
}
```

Koraci pri modeliranju u FLUENT-u su slični već opisanim u poglavlju 4.3.1. Jedina razlika je da umjesto unosa korisničke funkcije za (eng. *user defined function*) rubni uvjet stavljamo generirane profile sa meteorološkim podacima. Veličina vremenskog koraka je postavljena na 60 s. Broj vremenskih koraka je 2880. Nakon namještanja svih postavki, potrebno je pokrenuti journal file (*Read->Journal*) i rješavač FLUENT-a može početi računati zadani problem.

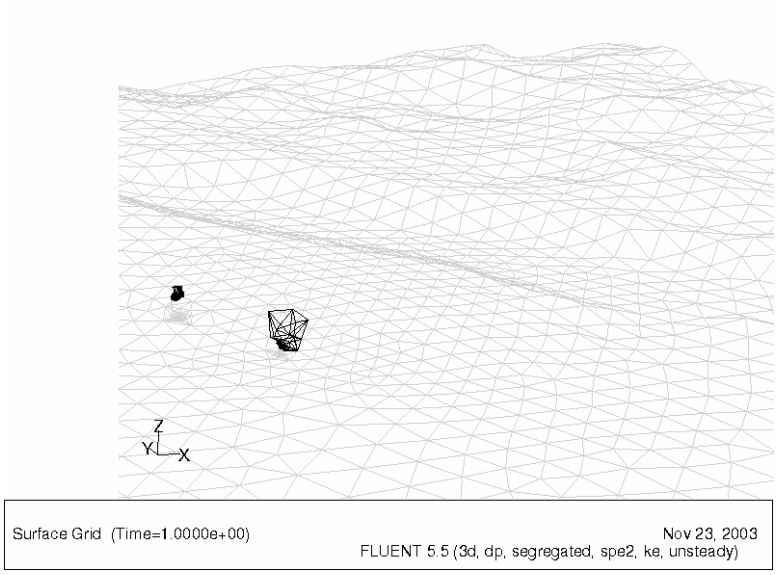
Nakon prođenih 5 vremenskih koraka, FLUENT više nije uspio konvergirati (Slika 4.3.9.). Pokušalo se smanjivanjem kriterija konvergencije, dužine vremenskog koraka, odabirom drugih podrelaksacijskih faktora utjecati na konvergenciju rješavača, ali nije bilo efekta.



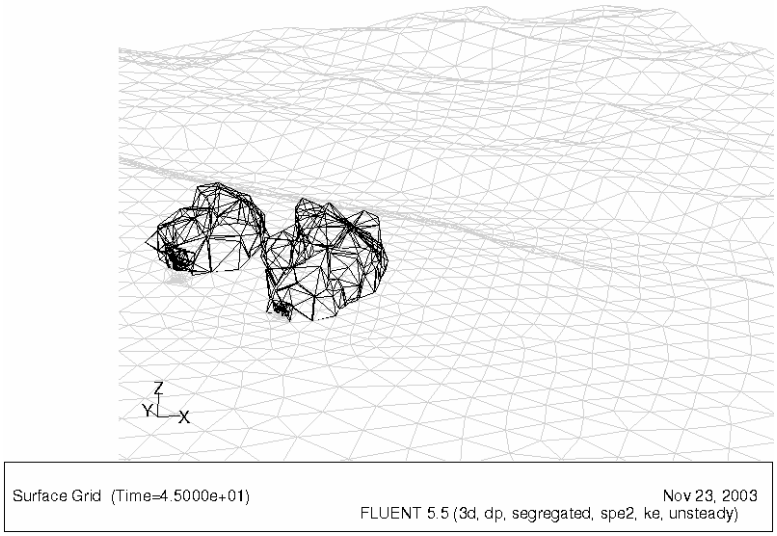
Slika 4.3.9. Prikaz prestanka konvergencije nakon petog vremenskog koraka

4.4. REZULTATI

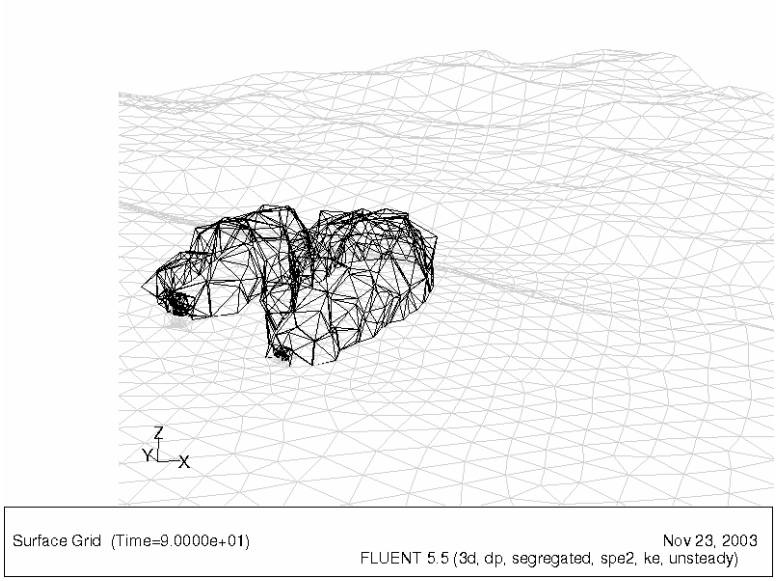
Sljedeće slike prikazuju raspršivanje sumpornog dioksida u vremeskom periodu od 360 s. Ploha koja izlazi iz dimnjaka predstavlja koncentraciju sumpornog dioksida od $1 \cdot 10^{-7}$ kg/m^3.



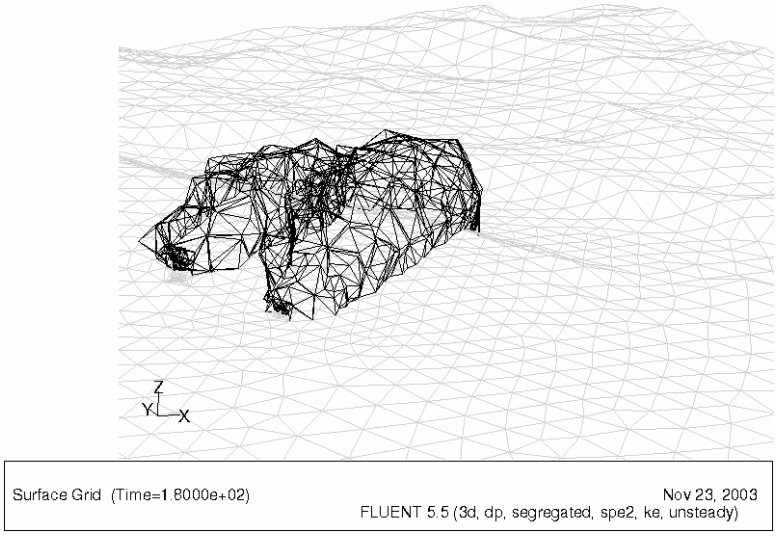
Slika 4.4.1. Prikaz perjanice SO_2 u 1-oj sekundi



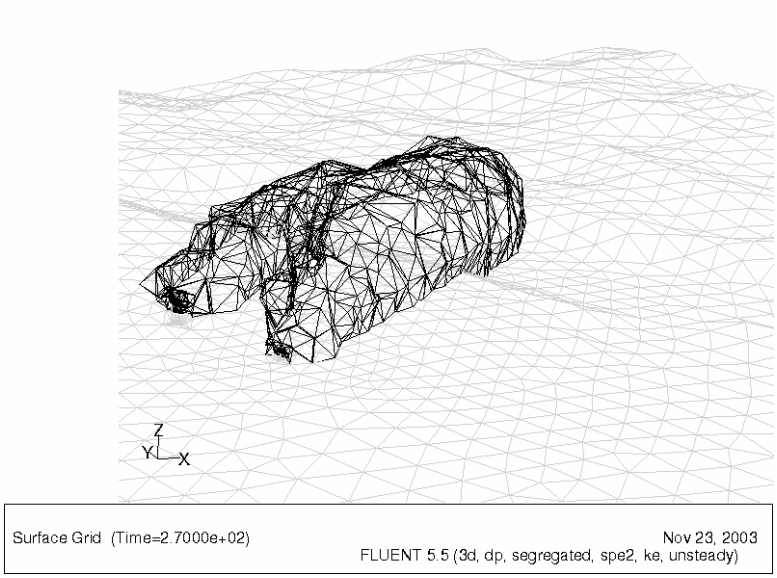
Slika 4.4.2. Prikaz perjanice SO_2 u 45-oj sekundi



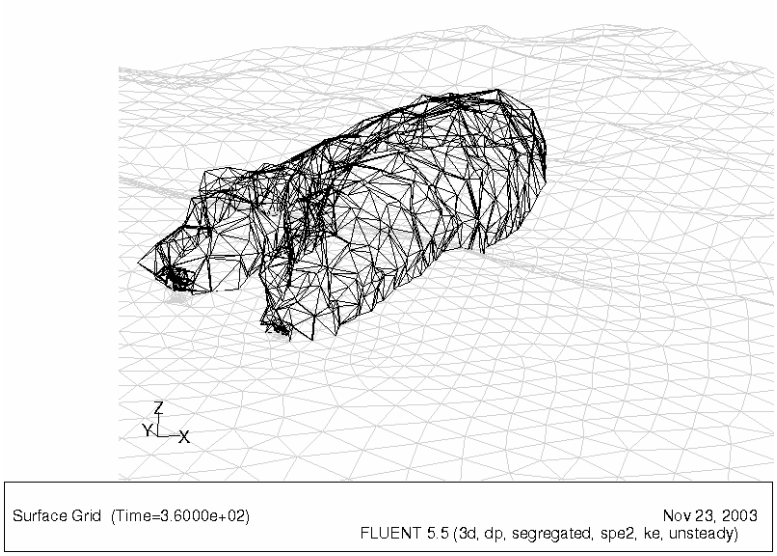
Slika 4.4.3. Prikaz perjanice SO_2 u 90-oj sekundi



Slika 4.4.4. Prikaz perjanice SO_2 u 180-oj sekundi



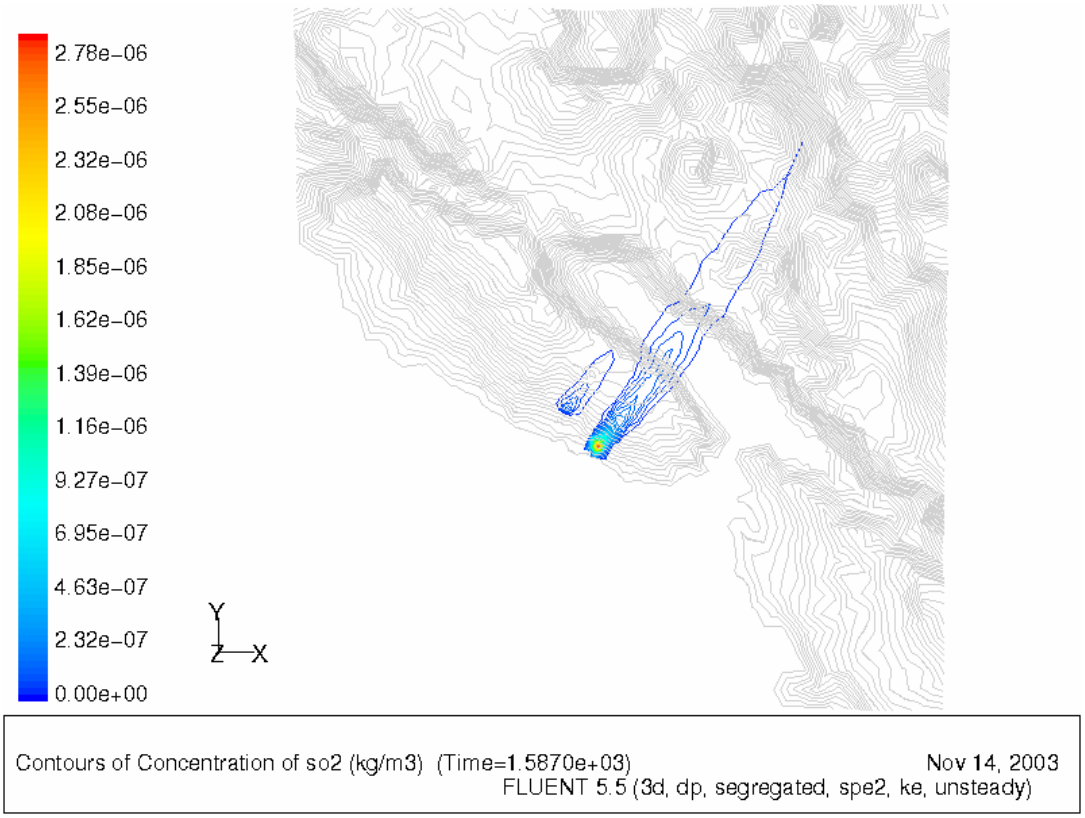
Slika 4.4.5. Prikaz perjanice SO_2 u 270-oj sekundi



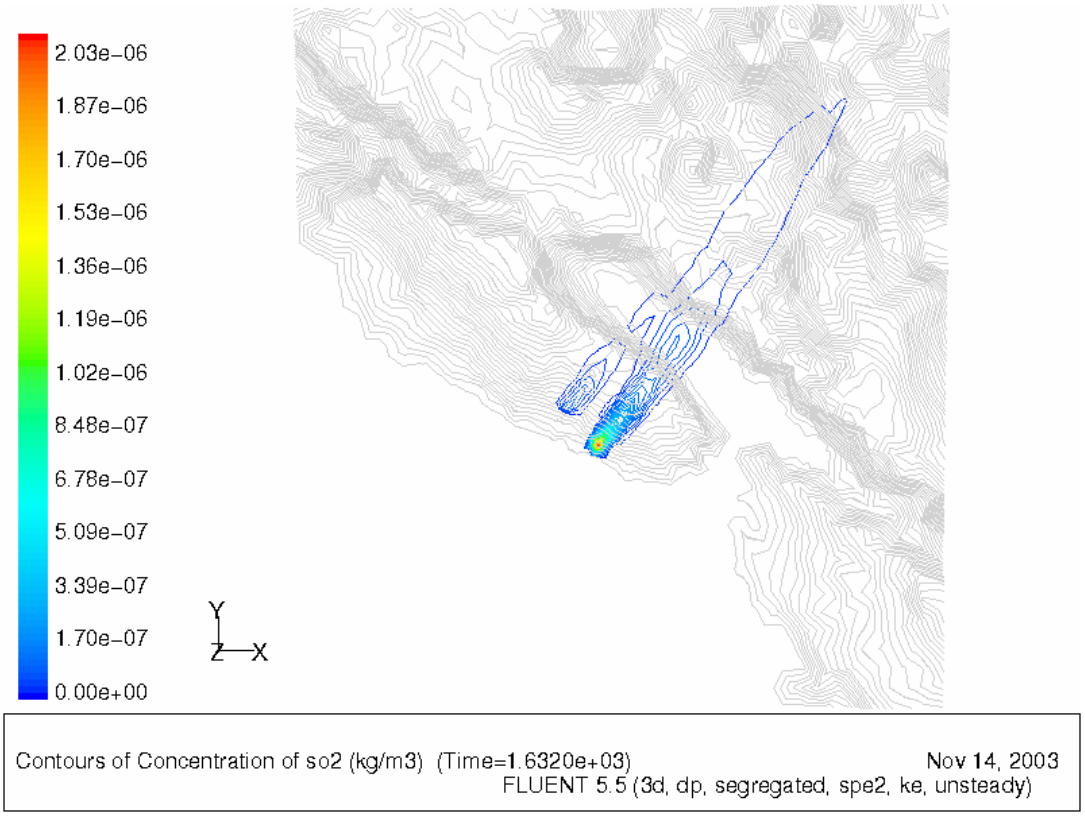
Slika 4.4.6. Prikaz perjanice SO_2 u 360-oj sekundi

Iz slika 4.4.1., 4.4.2., 4.4.3., 4.4.4., 4.4.5. i 4.4.6. vidljivo je da perjanica sumpornog dioksida prati konfiguraciju terena i zadani smjer vjetra.

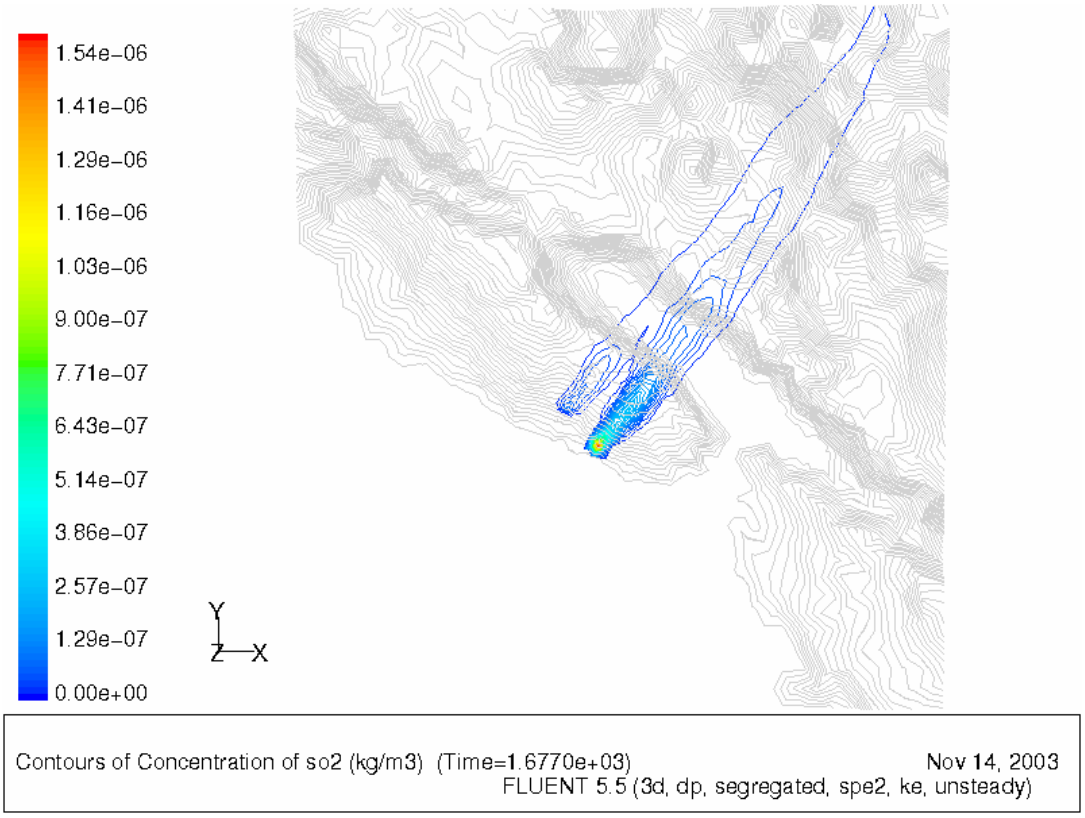
Sljedeće slike prikazuju koncentracije sumpornog dioksida na terenu.



Slika 4.4.7. Minimalna brzina vjetra 4,9 m/s



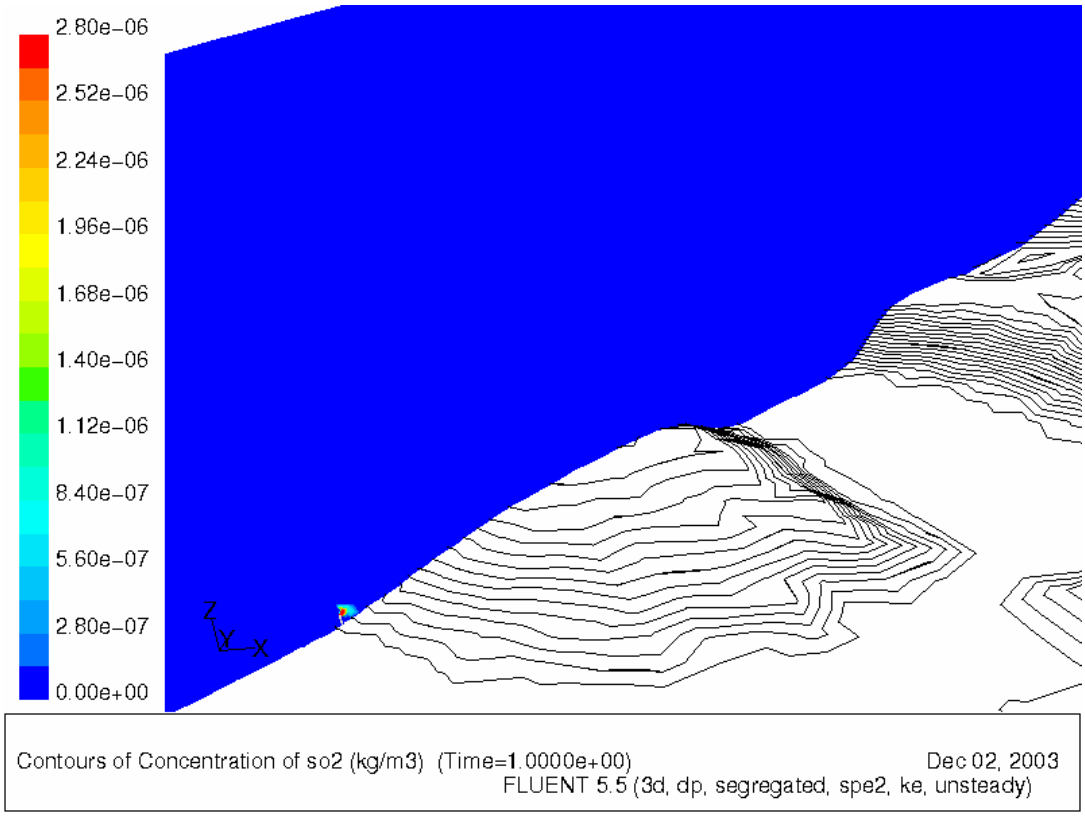
Slika 4.4.8. Srednja brzina vjetra 7 m/s



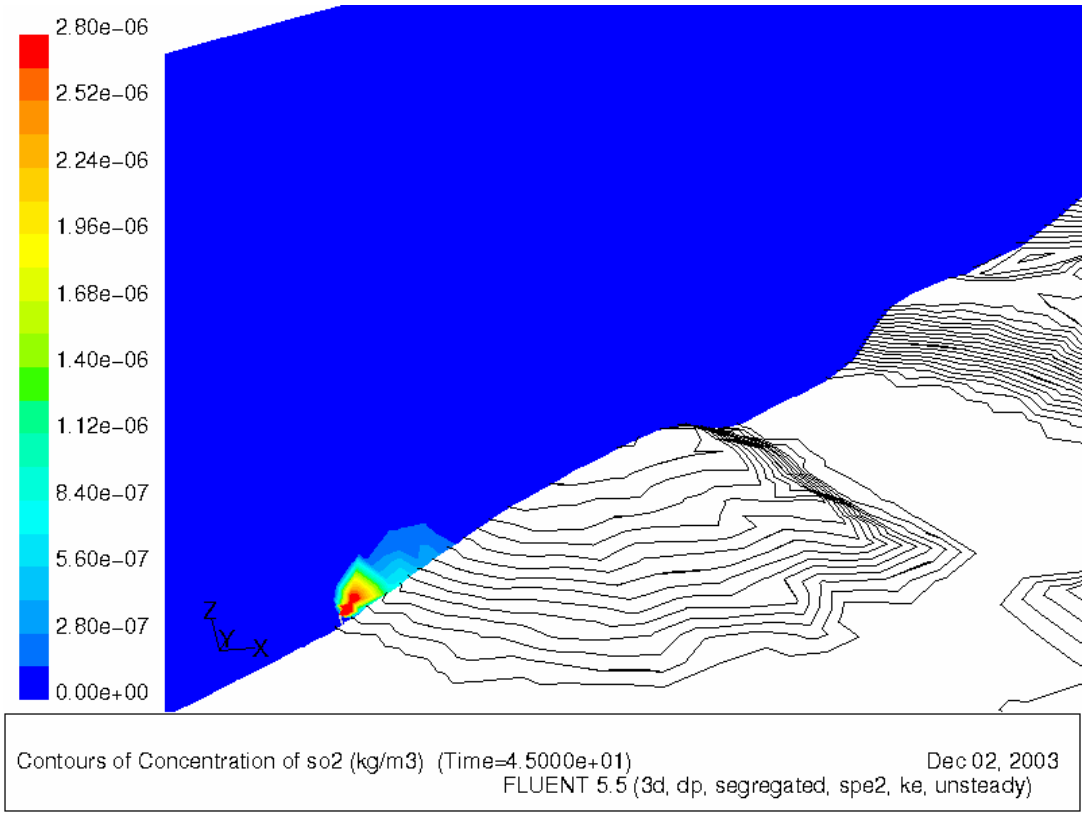
Slika 4.4.9. Maksimalna brzina vjetra 9,1 m/s

Na slikama 4.4.7., 4.4.8. i 4.4.9. vidljivo je da je prizemna koncentracija sumpornog dioksida manja pri većim brzinama vjetra, uz istovremeno povećavanje bočne disperzije

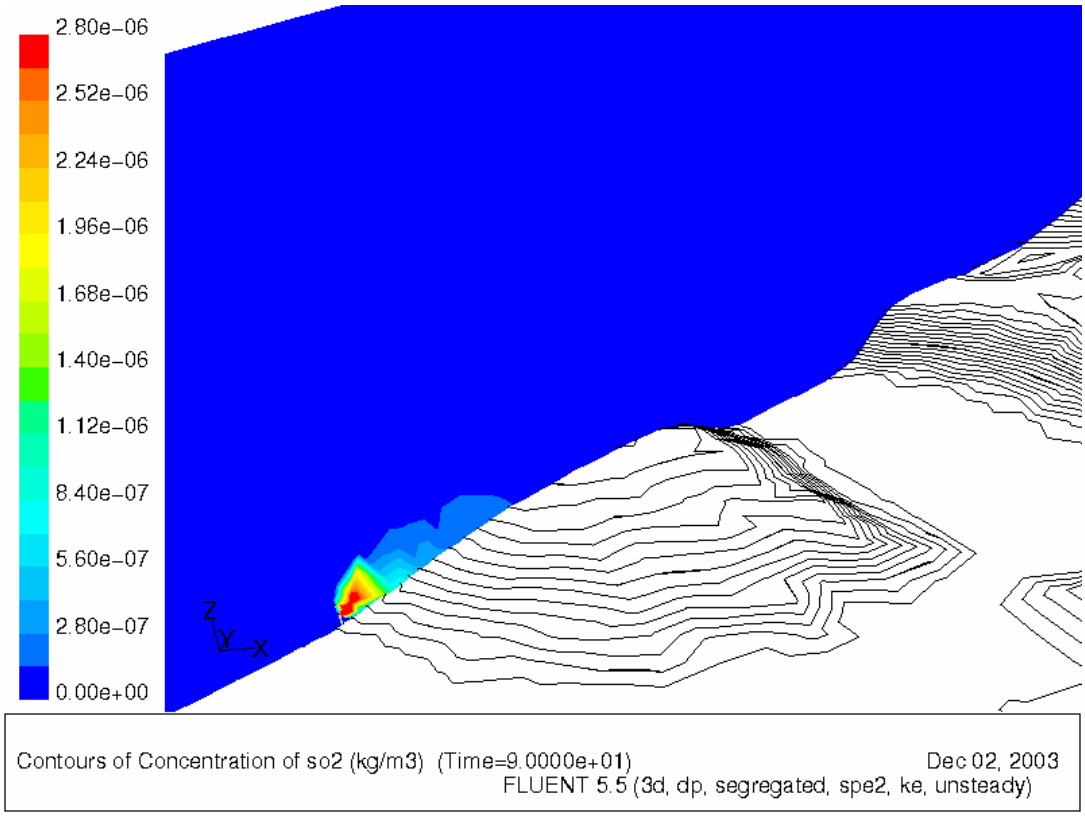
Sljedeće slike prikazuju koncentraciju sumpornog dioksida u ravnini okomitoj na teren koja prolazi kroz dimnjak i proteže se u smjeru strujanja vjetra (jugo-zapadni smjer vjetra).



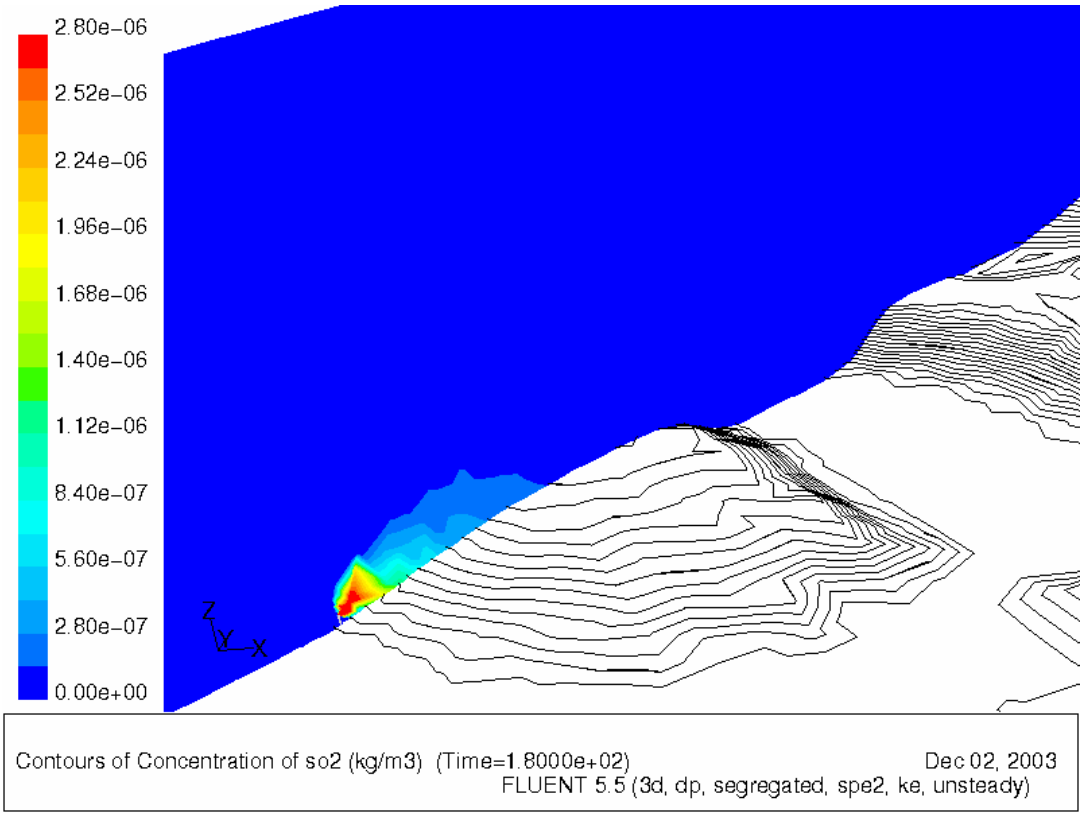
Slika 4.4.10. Prikaz koncentracije SO_2 u 1-oj sekundi



Slika 4.4.11. Prikaz koncentracije SO_2 u 45-oj sekundi



Slika 4.4.12. Prikaz koncentracije SO_2 u 90-oj sekundi



Slika 4.4.13. Prikaz koncentracije SO_2 u 180-oj sekundi

5. ZAKLJUČAK

Za ovakve vrste simulacije u FLUENT-u potrebno je iznimno jako računalo ili čak umreženo više računala da se u kraćem vremenu dobiju određeni rezultati. Budući da nisam imao takvo računalo, morao sam napraviti kompromis da bi mogao dovršiti diplomski rad u zadanom vremenu. Taj kompromis tiče se složenosti mreže nad modelom terena. Kako velika gustoća mreže nad modelom jako produžuje vrijeme rada, morao sam mrežu maksimalno pojednostaviti koliko je to moguće. Simulaciju dinamike perjanice sumpornog dioksida pokušao sam izvesti sa kompleksnim rubnim uvjetima (meteorološki podaci stanja atmosfere na rubnim stranicama modela) dobivenim iz meteorološkog programa MM5. Zamisao je bila da se rješavaču u FLUENT-u svaki vremenski korak ubacuju novi rubni uvjeti. Izgleda da je to bio presložen zadatak za FLUENT jer nije uspio doći do rješenja (nije konvergirao). Simulacija perjanice sumpornog dioksida sa rubnim uvjetima koje sam ja definirao (promjena brzine vjetra u obliku funkcije sinus) je uspjela i FLUENT je uspio dobiti rezultate koji su i prikazani u poglavlju 4.4. Prikazani rezultati su u skladu sa očekivanjima.

6. LITERATURA

1. FLUENT 5 User´s guide Volume 2, Fluent Incorporated, 1998. god.
2. FLUENT 5 User´s guide Volume 3, Fluent Incorporated, 1998. god.
3. Sonin, A. Ain, Equation of Motin for Viscous Fluids, Department of Mechanical Engineering, Massachusetts Institute of Technology, August 2001. god.
4. Schetz, A. Joseph, Injection and mixing in turbulent flow, American Istitute of Aeronautics and Astronautics, 1980. god.
5. Zhang, Xiaoming & Ghoinem, Ahmed F., A computer model for the rise and dispersion of winf-blown, buoyancy-driven plumes, United States Department of Commerce, Technology Administration, National Institute of Standards and Technology, NIST-GCR-93-637, 1993. god.
6. Hwang, Robert R., Numerical simulation turbulent jets in a crossflow of stratified fluids, Institute of Physics, Academia Sinica, Taipei, Taiwan
7. Čavrak M., Kompjutersko simuliranje difuzije i konvekcije onečišćenja zraka, Diplomski rad, Rijeka, 2001. god.
8. Socolofs, Lecture2: The Diffusion Equation-Mixing, Transport, and Transformation, Institute for Hydromechanics, University of Karlshrue, 2001. god.