ROZSZERZENIE OBIEKTOWO ZORIENTOWANEJ METODY ELEMENTÓW SKOŃCZONYCH O KONCEPCJĘ ELEMENTÓW SKOŃCZONYCH MODELOWANYCH ENERGETYCZNIE

Tadeusz WEGNER, Andrzej PĘCZAK*

* Instytut Mechaniki Stosowanej, Wydział budowy Maszyn i Zarządzania, Politechnika Poznańska, ul. Piotrowo 3, 60-965 Poznań

tadeusz.wegner@put.poznan.pl, andrzej.peczak@gmail.com

Streszczenie: W pracy opisano zastosowanie obiektowego programowania w języku C++ do metody elementów skończonych modelowanych energetycznie. Zaproponowano odporne na błędy i łatwe w stosowaniu rozwiązanie rozszerzające istniejące systemy MES o elementy modelowane energetycznie. Z wykorzystaniem funkcji energii właściwej odkształcenia oraz trójliniowych funkcji kształtu zbudowano trójwymiarowy ośmiowęzłowy element skończony typu „brick", a następnie opracowano model strukturalny umożliwiający wykorzystanie tego elementu w systemie MES. Przyjmując istniejące rozwiązania utworzono zbiór klas ułatwiający dostosowanie rozpatrywanego zagadnienia do istniejących kodów źródłowych. W pracy wykorzystano znane pojęcia takie jak klasy stopnia swobody, węzła i elementu. Bazując na wymienionych klasach, zaproponowano klasy pochodne, rozszerzone o nowe zmienne oraz metody. Analogicznie do rozwiązań znanych z obiektowej metody programowania elementów skończonych stosowanych do obliczania lokalnych macierzy, zaproponowano ogólną funkcję obliczającą wartość energii odkształcenia elementu. W związku z tym podjęto próbę uogólnienia funkcji obliczającej w kroku iteracyjnym nową pozycję węzła, w taki sposób, aby działała ona niezależnie od liczby i rodzaju elementów, do których węzeł przynależy, materiałów, które przyjęto w celu określenia mechanicznych właściwości elementów oraz stopni swobody, które węzeł posiada. Po wykazaniu łatwości i wygody stosowania zaproponowanego rozwiązania, zaprezentowano przykład numeryczny prostego modelu hiperelastycznego ciała poddanego odkształceniom.

1. WPROWADZENIE

W początkowym okresie implementacji systemów metod elementów skończonych wykorzystywano języki programowania proceduralno–strukturalnego. Wraz z rozwojem tych systemów wzrastał stopień skomplikowania kodu. Bardzo trudne stało się wyrażanie wzajemnych zależności pomiędzy funkcjami i danymi. Rozwój, modyfikacja i weryfikacja kodów były czasochłonne i kosztowne, ponieważ rozszerzanie systemu o nowe możliwości wymagało reorganizacji sporej ilości kodu. Słabością stał się brak efektywności i elastyczności w operowaniu danymi struktur. Zaczęto poszukiwać alternatywnych rozwiązań i sposobów programowania.

Programowanie obiektowe zaczęto stosować w metodzie elementów skończonych w latach 1985–1990. Różne hierarchie klas podstawowych modułów systemów MES proponowano w wielu pracach (Archer G. C. i inni , 1999; Besson J., Foerch R., 1997; Devloo P. R. B., 1997; Dolenc M., 2004; Kong. X. A. Chen D. P., 1995; Lichao Yo, Kumar A. V., 2001; Mackie R. I., 1990, 1997, 2001, 2004; Patzák B., Bittnar Z., 2001; Phongthanapanich S., Dechaumphai P., 2006; Rucki M. D., Miller G. R., 1998; Zimmerman Th. i inni, 1998).

Rozszerzenie systemu MES o nowe możliwości stało się łatwiejsze, bo opiera się na rozszerzeniu funkcjonalności klas już istniejących, a operowanie danymi tych klas dokonuje się za pomocą metod nowych lub starych.

Początkowo systemy MES w środowisku obiektowym implementowano w sposób bezpośredni. Później wyróżniono abstrakcyjne pojęcia takie jak stopień swobody, węzeł, element, materiał. Klasę stopnia swobody opisano w publikacjach (Balopoulos V., Abel J. F., 2002; Dubois–Pèlerin Y., Pegon P. 1998a; Mackie R. I., 2001). W klasie tej wyróżniono wartość stopnia swobody oraz przyjęte obciążenie. Węzeł przedstawiono w artykułach (Mackie R. I., 2001, 2004; Patzák B., Bittnar Z., 2001). Opis bazowej klasy elementu można znaleźć w pracach tych samych autorów (Mackie R. I., 1990, 1997, 2001, 2004; Patzák B., Bittnar Z., 2001).

Wraz z postępem czasu liczni autorzy coraz częściej stosowali techniki programowania obiektowego także do projektowania innych składowych elementów systemu MES. Chociaż większość autorów prac była najbardziej zainteresowana podstawami teorii MES, to jednak równolegle były prowadzone badania w zakresie generatorów siatek np. (Athanasiadis A. N., Deconinck H., 2003; Bastian M., Li B.Q., 2003; Ju J., Hosain M.U., 1996; Karamete B.K., 1997), interfejsów graficznych (Mackie R. I., 1990, 1997, 2001; Phongthanapanich S., Dechaumphai P., 2006; Zimmerman Th. i inni, 1998) czy też kontroli obliczeń (Mackie R. I. 1998, 2000, 2001, 2002). Okazało się, że programowanie systemu MES to nie tylko kilkadziesiąt procedur obliczeniowych wykorzystujących arytmetykę macierzową i że wymaga ono lepszego wykorzystania struktur danych oraz zarządzania nimi w prosty sposób. W każdej z tych dziedzin udowodniono, że programowanie obiektowe jest narzędziem bardzo ułatwiającym pracę.

W standardowej metodzie MES dla każdego elementu buduje się lokalną macierz sztywności. Standardowa klasa

elementu wymaga odpowiednich funkcji i zmiennych, dzięki którym możliwe jest zbudowanie jego macierzy sztywności. Po przetransformowaniu macierzy sztywności każdego z elementów do globalnego układu współrzędnych buduje się globalną macierzy sztywności, która wyraża związki pomiędzy siłami i przemieszczeniami. W celu znalezienia przemieszczeń węzłów trzeba rozwiązać układ równań liniowych. Wystarczy zatem tylko raz odwrócić globalną macierz sztywności, aby znaleźć potrzebne rozwiązanie. Rozwiązanie układu równań w programowaniu obiektowym dokonuje się za pomocą klas z odpowiednimi funkcjami realizującymi rozwiązanie określonego układu równań.

Obliczenia w nieliniowej mechanice ciał stałych wykonuje się poprzez wielokrotne odwracanie globalnej macierzy sztywności. Zadanie to przeprowadza się metodami iteracyjnymi, w których znajduje się kolejne przybliżenia położenia węzłów. W celu rozwiązania zagadnień nieliniowych, niektórzy autorzy zaproponowali rozszerzenie klas odpowiedzialnych za rozwiązywanie układów równań linowych, o nowe metody i zmienne, wykorzystując mechanizmy dziedziczenia i polimorfizmu (Commend S., Zimmerman T., 2001; Dubois–Pèlerin Y., Pegon P., 1998b; Lages E. N. i inni, 1999; Menétrey Ph., Zimmermann Th., 1993).

W pracy tej zaproponowano rozszerzenie istniejących systemów MES, bazujących na metodzie programowania obiektowego, o elementy skończone modelowane energetycznie za pomocą funkcji gęstości energii odkształcenia, wyrażającej energię odkształcenia materiału odniesioną do jednostki objętości ciała w stanie nieodkształconym, nazywaną energią właściwą odkształcenia. Oparto się na znanych wcześniej rozwiązaniach takich jak klasy stopnia swobody, węzła, elementu oraz materiału. Zaproponowano klasę elementu, która nie posiada danych ani funkcji reprezentujących macierz sztywności. W zamian zamieszczono procedury zwracające wartości energii odkształcenia elementu. Dodano również przykładowe metody operujące na węzłach elementu oraz zmienne umożliwiające dostęp do stałych materiałowych. W klasie węzła zaproponowano uniwersalną funkcję obliczającą jego nowe położenie.

2. FUNKCJA ENERGII ODKSZTAŁCENIA

Zachowanie nieliniowego elastycznego materiału opisuje się za pomocą funkcji energii odkształcenia. Funkcja energii odkształcenia materiału izotropowego zależy jedynie od stanu odkształcenia materiału. Ponieważ funkcja ta nie może zależeć od przyjętego układu współrzędnych, więc powinna być funkcją niezmienników stanu odkształcenia materiału, stąd

$$U = U(I_1, I_2, I_3). \tag{1}$$

Niezmienniki najczęściej cytowane w literaturze są powiązane z tensorem deformacji D za pomocą związków

$$I_1 = tr(D) = \lambda_1^2 + \lambda_2^2 + \lambda_3^2, \tag{2}$$

$$I_2 = \tfrac{1}{2}[tr(D)^2 - tr(D^2)] = \lambda_1^2\lambda_2^2 + \lambda_2^2\lambda_3^2 + \lambda_1^2\lambda_3^2, \tag{3}$$

$$I_3 = \det(D) = \lambda_1^2\lambda_2^2\lambda_3^2. \tag{4}$$

Można je interpretować (Wegner T. 1997) następująco: dla nieodkształconego sześcianu o ścianach prostopadłych do głównych kierunków odkształcenia i jednostkowej długości krawędzi, składowe odkształcenia λ_1^2, λ_2^2, λ_3^2 oznaczają kwadraty długości boków prostopadłościanu powstałego w wyniku odkształcenia tego sześcianu. Niezmiennik I_1 jest kwadratem długości przekątnej prostopadłościanu, I_2 jest sumą kwadratów powierzchni jego trzech ścian o wspólnym wierzchołku, a I_3 jest kwadratem jego objętości.

Zakładając liniową zależność pomiędzy naprężeniami i odkształceniami przy ścinaniu, Mooney (1940) w 1940 roku przyjął funkcję energii odkształcenia, opisującą zachowanie nieściśliwego materiału izotropowego, w postaci

$$U = C_1(I_1 - 3) + C_2(I_2 - 3). \tag{5}$$

W 1956 roku Rivlin (1956) zaproponował uogólnione sformułowanie wzoru (5) w postaci nieskończonej sumy składników zależnych od niezmienników tensora deformacji

$$U = \sum_{i+j+k=1}^{\infty} C_{ijk}(I_1 - 3)^i(I_2 - 3)^j(I_3 - 1)^k. \tag{6}$$

Jeżeli przyjmie się, że materiał jest nieściśliwy, to wzór opisujący tę cechę jest następujący

$$J = \sqrt{I_3} = \lambda_1 \lambda_2 \lambda_3 = 1. \tag{7}$$

Niezmienniki I_1 i I_2 nie zawierają informacji odnoszącej się wyłącznie do stanu odkształcenia czysto postaciowego, ponieważ dla deformacji czysto objętościowej, np. w stanie równomiernego trójosiowego rozciągania lub ściskania $\lambda_i = \lambda$ niezmienniki te są różne od zera i przyjmują wartości.

$$I_1^V = 3\lambda^2,\ I_2^V = 3\lambda^4,\ I_3^V = \lambda^6. \tag{8}$$

Taka niedogodność powodowała trudności w rozdzieleniu energii odkształcenia czysto postaciowego i czysto objętościowego na dwa odrębne składniki.

W swojej pracy Wegner (1997) zaproponował zmodyfikowane wielkości niezmienników z wyodrębnioną składową deformacji czysto objętościowej

$$I_i^S = I_i - I_i^V,\ i \in \{1, 2\}, \tag{9}$$

gdzie wykorzystując zależności (8) przyjął $I_3=\lambda^6$, co pozwoliło zapisać energię odkształcenia w postaci sumy energii odkształcenia czysto postaciowego $U^S(I_1^S,\ I_2^S)$ oraz energii odkształcenia czysto objętościowego $U^V(I_3)$

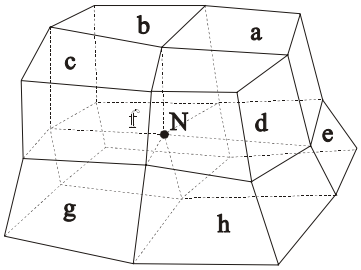
$$U(I_1^S, I_2^S, I_3) = U^S(I_1^S, I_2^S) + U^V(I_3). \tag{10}$$

3. ENERGETYCZNA METODA RELAKSACJI LOKALNEJ

Nieliniowe zagadnienia w metodzie elementów skończonych zazwyczaj rozwiązuje się iteracyjną metodą

Newtona–Raphsona. W arytmetyce macierzowej, na której opiera się klasyczna metoda elementów skończonych, po złożeniu lokalnych sztywności w globalną macierz i uwzględnieniu warunków brzegowych, w celu znalezienia przemieszczeń węzłów pod wpływem narastającej siły, należy wielokrotnie odwracać globalną macierz sztywności, ponieważ wraz z narastaniem obciążenia elementy macierzy sztywności ulegają zmianie.

Jedną z modyfikacji MES, wykorzystującą metodę stycznych Newtona–Raphsona, jest metoda relaksacji lokalnej przedstawiona w pracy Wegnera (1997). W zaproponowanej metodzie obliczeniowej zrezygnowano z określania globalnej macierzy sztywności na rzecz sztywności lokalnej. Właściwości fizyczne materiału określone są za pomocą funkcji energii właściwej odkształcenia opisującej energetyczny stan otoczenia w danym wybranym punkcie materiału. Oddziaływanie lokalne otoczenia na punkt należący do tego otoczenia – węzeł, opisuje przyjęty model materiału. Otoczenie lokalne węzła reprezentowane jest przez elementy skończone, które w odróżnieniu od elementów określonych w klasycznej metodzie elementów skończonych są modelowane energetycznie za pomocą wspomnianej wcześniej funkcji energii właściwej odkształcenia. Wykorzystując iteracyjną metodą Newtona–Raphsona poszukuje się takiej konfiguracji położenia węzłów, dla której znajdują się one w stanie równowagi. Położenie każdego węzła jest poprawiane jednorazowo w każdym kroku iteracji i ograniczone jest do obszaru elementów połączonych bezpośrednio z aktualnie obliczanym węzłem. Węzeł *N* jest wspólnym węzłem otaczających go elementów *a, b, c, d, e, f, g, h,* które tworzą fragment trójwymiarowej siatki. Fragment tej siatki pokazano na rysunku (rys. 1).



Rys. 1. Fragment siatki złożonej z 8 elementów

Energię odkształcenia takiego układu można zapisać jako sumę energii odkształcenia poszczególnych elementów

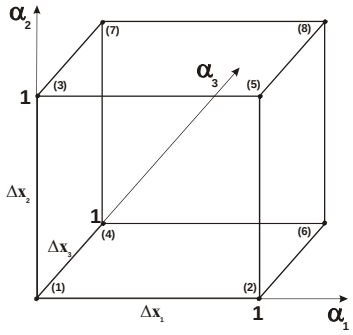
$$E = \sum_{k=1}^{8} E^{(k)}. \tag{11}$$

Przemieszczenia punktów zaproponowanego ośmiowęzłowego elementu (rys. 2) opisano przez trzy trójliniowe funkcje lokalnych współrzędnych α_i w postaci wzorów

$$\begin{aligned} u_i(\alpha_1,\alpha_2,\alpha_3) = a_0^i + a_1^i\alpha_1 + a_2^i\alpha_2 + a_3^i\alpha_3 + \\ a_{12}^i\alpha_1\alpha_2 + a_{13}^i\alpha_1\alpha_3 + a_{23}^i\alpha_2\alpha_3 + a_{123}^i\alpha_1\alpha_2\alpha_3, \end{aligned} \tag{12}$$

a pochodne cząstkowe określono jako

$$\frac{\partial u_i}{\partial x_i} = \frac{\partial u_i}{\partial \alpha_i}\frac{\partial \alpha_i}{\partial x_i}. \tag{13}$$



Rys. 2. 8 węzłowy element skończony w układzie lokalnych współrzędnych $\alpha_1, \alpha_2, \alpha_3$

Współczynniki przyjmują następującą postać

$$a_0^i = u_i^{(1)}, \tag{14}$$

$$a_1^i = u_i^{(2)} - u_i^{(1)}, \tag{15}$$

$$a_2^i = u_i^{(3)} - u_i^{(1)}, \tag{16}$$

$$a_3^i = u_i^{(4)} - u_i^{(1)}, \tag{17}$$

$$a_{12}^i = u_i^{(1)} - u_i^{(2)} + u_i^{(5)} - u_i^{(3)}, \tag{18}$$

$$a_{13}^i = u_i^{(1)} - u_i^{(2)} + u_i^{(6)} - u_i^{(4)}, \tag{19}$$

$$a_{23}^i = u_i^{(1)} - u_i^{(3)} + u_i^{(7)} - u_i^{(4)}, \tag{20}$$

$$\begin{aligned} a_{123}^i = u_i^{(2)} - u_i^{(1)} + u_i^{(3)} - u_i^{(5)} + \\ + u_i^{(4)} - u_i^{(6)} + u_i^{(8)} - u_i^{(7)}. \end{aligned} \tag{21}$$

W ten sposób składowe przemieszczenia wyrażono w sposób przybliżony, w zależności od przemieszczenia węzłów elementu. Współrzędne punktów elementu w globalnym układzie opisane są następująco

$$x_i = x_i^{(1)} + \alpha_i \Delta x_i. \tag{22}$$

Tensor odkształceń wyraża się za pomocą

$$\Lambda = \begin{bmatrix} \lambda_{xx} & \lambda_{xy} & \lambda_{xz} \\ \lambda_{yx} & \lambda_{yy} & \lambda_{yz} \\ \lambda_{zx} & \lambda_{zy} & \lambda_{zz} \end{bmatrix}, \tag{23}$$

gdzie

$$\lambda_{xx} = 1 + \frac{\partial u}{\partial x},\ \lambda_{xy} = \frac{\partial u}{\partial y},\ \lambda_{xz} = \frac{\partial u}{\partial z}, \tag{24}$$

$$\lambda_{yx} = \frac{\partial v}{\partial x},\ \lambda_{yy} = 1 + \frac{\partial v}{\partial y},\ \lambda_{yz} = \frac{\partial v}{\partial z}, \tag{25}$$

$$\lambda_{zx} = \frac{\partial w}{\partial x},\ \lambda_{zy} = \frac{\partial w}{\partial y},\ \lambda_{zz} = 1 + \frac{\partial w}{\partial z}. \tag{26}$$

Tensor deformacji *D* ma zatem postać

$$D = \Lambda^T \Lambda . \tag{27}$$

Podstawiając zależności (2–4) do wzoru opisującego funkcję energii właściwej zależną od niezmienników, otrzymuje się wyrażenie na energię odkształcenia U zależną od składowych przemieszczenia u_i węzłów oraz lokalnych współrzędnych α_i.

$$U(u_i^{(1)}, u_i^{(2)}, u_i^{(3)}, u_i^{(4)}, u_i^{(5)}, u_i^{(6)}, u_i^{(7)}, u_i^{(8)}; \alpha_i). \tag{28}$$

Całkując energię właściwą względem objętości otrzymuje się energię odkształcenia, którą opisuje wzór

$$E = \int_V U dV , \tag{29}$$

gdzie V oznacza objętość elementu w stanie nieodkształconym.

Po podzieleniu elementu na podelementy i zastąpieniu całki przez sumę energii odkształcenia tych podelementów, otrzymuje się wyrażenie na energię odkształcenia elementu

$$E = \prod_{i=1}^{3} \frac{\Delta x_i}{l_e^{(i)}} \sum_{l^{(1)}=1}^{l_e^{(1)}} \sum_{l^{(2)}=1}^{l_e^{(2)}} \sum_{l^{(3)}=1}^{l_e^{(3)}} U(l^{(1)}, l^{(2)}, l^{(3)}), \tag{30}$$

gdzie

$$U(l^{(1)}, l^{(2)}, l^{(3)}) = U(\alpha_1(l^{1)}), \alpha_2(l^{(2)}), \alpha_3(l^{(3)})), \tag{31}$$

oznacza energię właściwą odkształcenia zależną od lokalnych współrzędnych $\alpha_i(l^{(i)})$ które oznaczają punkty elementu, w których oblicza się wartości energii właściwej. Wyrażone są one za pomocą

$$\alpha_i(l^{(i)}) = \frac{l^{(i)} - \frac{1}{2}}{l_e^{(i)}}, \; l^{(i)} \in \{1, 2, ..., l_e^{(i)}\}, \tag{32}$$

gdzie $l_e^{(i)}$ oznaczają liczby podziału elementu na podelementy odpowiednio w kierunkach współrzędnych α_i, przy założeniu, że energia właściwa każdego punktu podelementu jest w przybliżeniu równa energii właściwej punktu leżącego w jego środku.

Położenia węzłów odpowiadające stanowi równowagi określa się w kolejnych krokach iteracyjnych odpowiedniej procedury. Wyznaczanie położenia węzła w jednym kroku iteracyjnym jest ograniczone do obszaru ośmiu elementów, dla których węzeł ten jest wspólny. W każdym kroku iteracji poprawia się położenie tylko jednego węzła względem pozostałych. Każdy węzeł ma trzy stopnie swobody. Położenie węzła jest poprawiane w taki sposób, aby energia układu elementów była jak najmniejsza. Jego nowe położenie jest określane przy pomocy lokalnej sztywności obliczonej oddzielnie dla każdego stopnia swobody węzła. Takie postępowanie wykonuje się wielokrotnie dla wszystkich węzłów, czego wynikiem jest minimalizacja energii odkształcenia układu. Obliczenia są wykonywane aż do uzyskania stanu odkształcenia, dla którego dokładność aproksymacji można uznać za wystarczającą.

Rozpoczynając obliczenia należy najpierw przesunąć wspólny węzeł N (rys. 1) o dowolną, małą wartość Δu_i, w dodatnim oraz ujemnym kierunku zgodnym ze stopniem swobody

$$E_0(u_i + \Delta u_i) = \sum_{j=1}^{8} E^{(j)}(u_i^j + \Delta u_i), \tag{33}$$

$$E_0(u_i - \Delta u_i) = \sum_{j=1}^{8} E^{(j)}(u_i^j - \Delta u_i). \tag{34}$$

Znając wartość przemieszczenia węzła oraz wartość przyrostu energii elementów otaczających węzeł, można obliczyć prawo– i lewostronne wartości oddziaływania sił składowych węzła na lokalne otoczenie

$$F_P(u_i) = \frac{E_0(u_i + \Delta u_i) - E_0}{\Delta u_i}, \tag{35}$$

$$F_L(u_i) = \frac{E_0 - E_0(u_i - \Delta u_i)}{\Delta u_i}. \tag{36}$$

Następnie należy wyznaczyć średnią arytmetyczną wartości sił składowych oddziaływania węzła

$$F(u_i) = \frac{F_P(u_i) + F_L(u_i)}{2} \tag{37}$$

oraz dodatkowo wartości lokalnych sztywności odpowiadających danemu stopniowi swobody. Lokalną sztywność określa się poprzez różnicę siły prawo oraz lewostronnej podzieloną przez wartość przemieszczenia węzła

$$k(u_i) = \frac{F_P(u_i) - F_L(u_i)}{\Delta u_i}. \tag{38}$$

Uwzględniając działającą na węzeł zewnętrzną siłę S, składowe wypadkowej siły R działającej na węzeł opisane są wzorami

$$R(u_i) = S(u_i) - F(u_i). \tag{39}$$

Dla węzła w położeniu równowagi siła wypadkowa powinna być równa zero. Położenie równowagi węzła w kroku iteracyjnym wyznaczamy z zależności

$$u_i^w = u_i^{w-1} + \frac{R(u_i)}{k(u_i)}. \tag{40}$$

We wzorach tych indeks w oznacza numer kolejnej iteracji, natomiast indeks i oznacza kierunek zgodny ze stopniami swobody.

4. PROGRAMOWANIE OBIEKTOWE – HIERARCHIA KLAS

4.1 Materiał i energia właściwa odkształcenia

Mechaniczne właściwości materiału opisuje klasa *TMaterial*.

```
class TMaterial
{
 TStrainEnergy *f_strain_energy;
public:
```

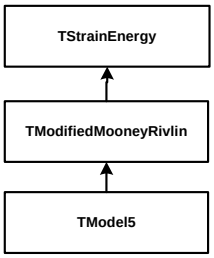
```
 double Kirchoff;
 double Poison;
 double Young;
 double *C;
 int C_size;

 TMaterial();
 TMaterial(TStrainEnergy *);
 virtual ~TMaterial();
 TStrainEnergy *get_strain_energy() const;
 void set_strain_energy(TStrainEnergy *);
};
```

Zawiera ona podstawowe składowe – parametry i funkcje – opisujące mechaniczne właściwości materiału. Ponieważ materiał, który będziemy używali jest materiałem o charakterystyce nieliniowej, opisywanym przez funkcję energii odkształcenia, to w klasie *TMaterial* zamieściliśmy składową o nazwie *f_strain_energy,* która jest typu *TStrainEnergy**. Jest ona klasą abstrakcyjną zdefiniowaną jako kontener funkcji, które reprezentują ogólny zbiór modeli funkcji energii właściwych odkształcenia. Rozważania na temat tej klasy opisali Jeremić B. i inni (1999).

```
class TStrainEnergy
{
protected:
 TMaterial *f_parent_material;
public:
 TStrainEnergy();
 TStrainEnergy(TMaterial *);
 virtual ~TStrainEnergy();
 TMaterial *get_parent_material() const;
 void set_parent_material(TMaterial *);
 virtual double W(const TDeformationTensor &) const=NULL;
 virtual double dev_W(const TDeformationTensor &)
 const=NULL;
 virtual double vol_W(const TDeformationTensor &)
 const=NULL;
};
```

Zadeklarowane w tej klasie funkcje *W(TdeformationTensor &), dev_W(TDeformationTensor &)* oraz *vol_W (TDeformationTensor &)* są wirtualne i przyjmują argument w postaci tensora deformacji, a zwracają w odpowiedniej kolejności wartości energii właściwej całkowitej, odkształcenia postaciowego i odkształcenia objętościowego, które zależą od wartości elementów tensora odkształcenia. Z klasy *TStrainEnergy* należy wydziedziczać klasy, które będą w bezpośredni lub pośredni sposób definiowały modele energii właściwych odkształcenia dla interesujących nas materiałów. Klasa *TModifiedMooneyRivlin* jest klasą abstrakcyjną i została wprowadzona wyłącznie z powodów formalnych. Reprezentuje ona zbiór zmodyfikowanych energii odkształcenia Mooneya–Rivlina opisanych w pracy Wegnera (1997). Z klasy tej wydziedziczono klasę pochodną, która reprezentuje użyty w obliczeniach model funkcji energii właściwej i jest reprezentowany przez klasę *TModel5*.



Rys. 3. Hierarchia klas reprezentujących model energii odkształcenia

Pole *parent_material* zawarte w klasie *TStrainEnergy* jest wypełniane w jej konstruktorze lub poprzez metodę *set_parent_material(TMaterial*)*. Zapewnia ono dostęp obiektom typu *TStrainEnergy* do stałych materiałowych.

4.2 Element

Klasy stopnia swobody, węzła oraz elementu pierwszy wprowadził w swoich pracach Zimmerman i inni (1991, 1992a, 1992b, 1993).

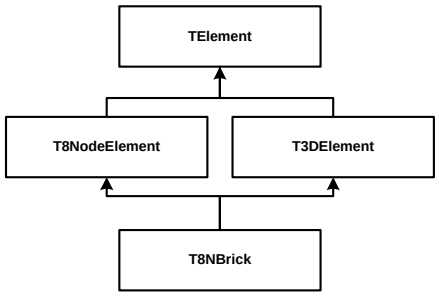
Element reprezentuje mechaniczne właściwości materiału, który opisuje funkcja energii właściwej odkształcenia oraz jest odpowiedzialny za relacje pomiędzy węzłami (i ich stopniami swobody), które z nim stowarzyszono. Klasa *TEnergElement* jest klasą bazową wszelkich klas reprezentujących dowolnego rodzaju energetyczne elementy skończone. W klasie tej zrezygnowano z metod służących do budowania macierzy sztywności pojedynczego elementu, natomiast wprowadzono metodę obliczającą jego energię właściwą oraz energię odkształcenia. Definicja tej klasy jest następująca

```
class TEnergElement
{
protected:
 unsigned int f_dimension;
 unsigned int f_subelements;
 TMaterial *f_material;
 TExtNodes *f_nodes;
 vector<TIntegrationPoint> *f_integration_points;
 TDeformationTensor f_deformation_tensor;
 virtual void precalculate_integration_points();
public:
 TEnergElement ();
 TEnergElement (TMaterial *) ;
 virtual ~TEnergElement ();
 TMaterial *get_material() const;
 void set_material(TMaterial *);
 virtual void set_subelements(unsigned int);
 int add_node(TNode *);
 void delete_node(int);
 void remove_node(TNode *);
 int node_index();
 TExtNode *get_node(unsigned int) const ;
 bool set_node(TNode *, unsigned int);
 TExtNodes *get_nodes() ;
 virtual double energy() = NULL;
 double strain_energy();
 virtual void calculate_shape_funct_coeff();
 virtual TDeformationTensor &calculate_deformation_ten-
 sor_at(TIntegrationPoint &);
};
```

Ponieważ każdy z elementów reprezentuje fragment materiału, z którego jest wykonany, to nieodłączną składową klasy elementu jest zmienna *f_material*, która jest wskaźnikiem na typ opisujący materiał. Dostęp do tego pola zapewniają funkcje *get_material()* oraz *set_material(TMaterial*)*, które w odpowiedniej kolejności zwracają oraz ustawiają materiał elementu. Wymiar elementu określa *f_dimension*, a liczbę jego podelementów zmienna *f_subelements*. Liczbę podelementów ustala się za pomocą funkcji *set_subelements(unsigned int)*.

Każdy element posiada węzły. Składowa *f_nodes* jest listą węzłów, do których dostęp uzyskuje się za pomocą procedury *get_nodes(void)*. Aby stowarzyszyć węzeł z elementem wykorzystuje się procedurę *add_node(TNode*)*, przyjmującą jako argument węzeł, który ma być z elementem stowarzyszony. Do usunięcia węzła z elementu używa się procedur *delete_node(int)* lub *remove_node(TNode*)*. Argumentem funkcji *delete_node(int)* jest numer węzła, natomiast argumentem funkcji *remove_node(TNode *)* jest instancja węzła, który chcemy z elementu usunąć, Metodą *node_index(TNode*)* zwraca się indeks wybranego węzła stowarzyszonego z elementem. Funkcja *set_node(TNode*, unsigned int)* wstawia węzeł na określoną pozycję w liście węzłów elementu, *get_node(unsigned int)* zwraca węzeł z określonej pozycji w liście.

Całkowitą energię właściwą odkształcenia elementu określa się za pomocą deklaracji *strain_energy()*. Ponieważ w tej klasie nie określa się jakiego rodzaju element będzie wykorzystany, ile będzie posiadał węzłów oraz ile stopni swobody będą posiadały węzły, to nie można określić jakiego rodzaju funkcje kształtu zostaną użyte do opisania przemieszczeń węzłów elementu. Powoduje to niemożliwość określenia funkcji energii właściwej, która zależy bezpośrednio od przemieszczeń węzłów, a następnie energii odkształcenia, którą będziemy obliczać za pomocą funkcji *energy()*. Deklaracja funkcji *energy()* jako czystej metody wirtualnej zabezpiecza przed nieumyślnym utworzeniem instancji klasy *TEnergElement*, ponieważ byłaby ona bezużyteczna ze względu na brak definicji funkcji energii odkształcenia. Zadanie dostarczenia definicji tej funkcji należy do projektanta, który musi zaimplementować ją w klasie potomnej projektując potrzebny element.



Rys. 4. Hierarchia klas reprezentujących modele elementów skończonych

Z klasy *TEnergElement* wyprowadzono dwie klasy potomne: *T3DElement*, *T8NodeElement* (rys. 4). Pierwsza z nich reprezentuje elementy przestrzenne, a druga elementy z ośmioma węzłami. Klasy te są klasami abstrakcyjnymi i nie można tworzyć ich instancji. Następnie korzystając z dziedziczenia wielokrotnego utworzono klasę *T8NBrick* reprezentującą ośmiowęzłowe elementy przestrzenne typu „brick" (równoległoboki).

Klasa ta jest reprezentacją trójwymiarowego elementu przedstawionego na rysunku (rys. 2). Zazwyczaj w literaturze opisane są hierarchie klas elementów z wykorzystaniem dziedziczenia pojedynczego. Zapewne jednym z powodów takiego przybliżenia jest niedostępność mechanizmu wielokrotnego dziedziczenia w innych językach. Dzięki mechanizmowi wielokrotnego dziedziczenia łatwiejsze jest unikanie powielania kodu, jednak należy się nim posługiwać z należytą uwagą i ostrożnością.

Definicja klasy *T8NBrick* jest następująca:

```
class T8NBrick : public T3DElement, T8NodeElement
{
 double *f_delta_x;
public:
 TBrick ();
 TBrick (TNodes *);
 ~TBrick ();
 double energy(void);
};
```

Funkcja obliczająca energię właściwą elementu została zadeklarowana w klasie *TEnergElement* w następujący sposób

```
double TEnergElement::strain_energy()
{
 double EnergyValue=0.0;
 calculate_shape_funct_coeff();
 for(unsigned int i=0; i<f_integration_points->size(); i++){
  TIntegrationPoint &ip=(*f_integration_points)[i];
  TDeformationTensor &D=calculate_deformation_tensor_at(ip);
  EnergyValue+=(f_material->get_strain_energy())->W(D);
 }
 return EnergyValue;
}
```

Najpierw procedura *strain_energy()* oblicza wartości współczynników funkcjii kształtu za pomocą funkcji *calculate_shape_funct_coeff()*. Następnie, z wykorzystaniem tensora deformacji obliczanego w każdym wybranym punkcie elementu, sumuje wartości energii właściwej i na końcu wartość ta jest zwracana. Oczywiście punkty, w których funkcja oblicza wartości energii właściwej trzeba uprzednio wyliczyć, np. w konstruktorze klasy elementu. Jedyną czynnością jaką należy wykonać, aby w prawidłowy sposób funkcję tę wykorzystać w klasach elementów innego typu, jest przedefiniowanie procedur obliczających współczynniki funkcji kształtów oraz tensora deformacji, w taki sposób, aby były one zgodne z rodzajem stosowanego elementu skończonego. Zaletami tej funkcji są szybkość działania, zwięzłość, czytelność i uniwersalność, bo funkcja jest wspólna dla innych klas elementów energetycznych i można dzięki niej obliczać energię właściwą dowolnego elementu.

W klasie *T8NBrick* przedefiniowano funkcję *energy()* odpowiedzialną za obliczanie energii odkształcenia elementu. Jak widać obliczenie energii odkształcenia elementu jest niezwykle proste i sprowadza się wpierw do

obliczenia energii właściwej elementu dzięki wywołaniu metody *strain_energy()* z bazowej klasy *TEnergElement*, a następnie do przemnożenia przez odpowiednią wartość członu wyrażenia (30).

```
double T8NBrick::energy()
{
 double EnergyValue = TEnergElement ::strain_energy();
 for(unsigned int i=0; i<f_dimension; i++)
  EnergyValue *=(f_delta_x[i]/f_subelements);
 return EnergyValue;
}
```

4.3 Stopień swobody

Standardowa klasa *TDOF* reprezentuje stopnie swobody i zamyka w obrębie klasy powiązane z nimi dane takie jak przemieszczenia i siły zewnętrzne, w postaci danych nie powiązanych ze współrzędnymi geometrycznymi. Zmienna *f_value* zawiera wartość stopnia swobody, czyli w naszym przybliżeniu jest to wartość przemieszczenia, a zmienna *f_saved_value* przechowuje wartość przemieszczenia obliczonego w poprzednim kroku iteracyjnym. Składowa *f_load* jest składową obciążenia czyli siłą zewnętrzną przyłożoną do węzła. Zapis wartości do tych zmiennych realizuje się odpowiednio przez funkcje *set_value(double)*, *save_value()* oraz *set_load(double)*, natomiast zmienne te można odczytać za pomocą funkcji *get_value(), get_saved_value()* i *get_load()*. Procedura *add_load(double)* dodaje wartość obciążenia do składowej siły zewnętrznej.

```
class TDOF
{
 double f_value;
 double f_saved_value;
 double f_load;
public:
 TDOF();
 virtual ~TDOF();
 void add_load(double) ;
 double get_load() const;
 void set_load(double);
 double get_value() const;
 void set_value(double);
 void save_value();
 double get_saved_value() const;
};
```

Z wykorzystaniem klasy *TDOF* utworzono klasę *TExtDOF*. Klasa ta jest rozszerzona o nowe funkcje i zmienne wykorzystywane w rozpatrywanym przez nas zagadnieniu. Procedura *void apply_constrain(bool)* utwierdza lub uwalnia z więzów wybrany stopień swobody, a funkcja *is_constrained()* zwraca wartość informującą, czy wybrany stopień swobody jest utwierdzony, czy też nie. W naszym przybliżeniu te dwie procedury wykorzystano wykonując obliczenia nowych pozycji wybranych stopni swobody węzła.

```
class TExtDOF : public TDOF
{
 double f_delta;
 bool f_constrained;
public:
 TExtDOF();
 virtual ~TExtDOF();
 void apply_constrain(bool);
 bool is_constrained() const;
 double get_delta() const;
 void set_delta(double);
};
```

4.4 Węzeł

Klasa *TNode* reprezentuje węzeł, który posiada pewne współrzędne geometryczne oraz stopnie swobody. Z wykorzystaniem dziedziczenia klasę tą utworzono z klasy *TPoint*. Standardowo w klasie węzła zawarty jest zbiór stopni swobody. Stopnie swobody węzła przechowywane są w liście *f_dofs*, a dostęp do węzłów uzyskuje się za pomocą funkcji *get_dof_list()*. Liczbę stopni swobody można ustawić wywołując funkcję *set_dof_number(int)*, a uzyskać za pomocą funkcji *get_dof_number()*. Instancję klasy węzła zazwyczaj tworzy się podając współrzędne węzła w konstruktorze klasy. Każdy węzeł należy do jednego lub kilku elementów skończonych. Obiekty je reprezentujące umieszcza się w indeksowanej liście o nazwie *f_elements*, która jest składową klasy *TNode*. Dostęp do listy elementów węzła zapewnia procedura *get_elements(void)*. Aby dodać element do listy elementów węzła należy wywołać procedurę *add_element(TEnergElement*)*, która zwraca numer węzła po dodaniu do elementu. Do usunięcia elementu z listy elementów służą procedury *delete_element(int)* oraz *remove_element(TEnergElement *)*. Nie ma powodu, dla którego procedury te powinny znajdować się w sekcji publicznej, gdyż elementy tworzy się wykorzystując już istniejące węzły, ponieważ to one opisują jego geometryczny kształt i położenie. Zadaniem elementu jest przekazanie informacji węzłowi, że będzie z nim stowarzyszony. W podobny sposób element musi poinformować węzeł, że nie będzie on już dłużej z nim stowarzyszony. Osiąga się to wywołując procedury do dodawania i usuwania węzła. Procedura *get_element(int)* udostępnia wybrany element węzła, a procedura *element_index(TEnergElement*)* zwraca jego numer. Inne informacje na temat klasy węzła można znaleźć w literaturze (Mackie R. I., 2001).

```
class TNode : public TPoint
{
 TEnergElements *f_elements;
 vector<TDOF *> *f_dofs;
protected:
 void delete_dofs();
public:
 TNode();
 TNode(double x, double y, double z);
 TNode(TPoint *Point);
 virtual ~TNode();
 void add_element(TEnergElement *);
 void delete_element(int);
 void remove_element(TEnergElement *);
 vector<TDOF *> *get_dof_list() const;
 virtual int get_dof_number() const;
 virtual void set_dof_number(int);
 TEnergElements *get_elements() const;
 TEnergElement *get_element(int) const;
};
```

Z klasy *TNode* wydziedziczono rozszerzoną klasę węzła *TExtNode*, w której przedefiniowano zmienną *f_dofs*

w taki sposób, aby możliwy był dostęp do rozszerzonych węzłów, którymi będziemy się posługiwali. Cel ten osiągnięto przeciążając operator indeksowania w klasie typu *list_of_extended_nodes<TDOF *>*. Rozwiązanie to zwalnia programistę z każdorazowego rzutowania typu *TDOF** na typ *TExtDOF **.

```
class TExtNode : public TNode
{
 double *f_right_side_force;
 double *f_left_side_force;
 list_of_extended_dofs<TDOF *> *f_dofs;
public:
 TExtNode();
 TExtNode(double x, double y, double z);
 TExtNode(TPoint *);
 virtual ~TExtNode();
 list_of_extended_dofs<TDOF *> * get_dof_list() const;
 double calculate();
 virtual int get_dof_number();
 virtual void set_dof_number(int);
};
```

W klasie *TExtNode* zadeklarowano funkcję *calculate(void)*. Jest to procedura, dzięki której węzeł samodzielnie koryguje swoją pozycję wykorzystując uprzednio opisany algorytm. Funkcja ta oblicza nowe wartości przemieszczenia węzła. Przykładowa definicja procedury obliczającej nowe położenie węzła może wyglądać tak

```
void TExtNode::calculate()
{
unsigned int i,j;
double F,k,ResidualForce,energy, last_energy = 0.0;
TEnergElements *Elements = get_elements();
TEnergElement *pElement;
TExtDOF *dof;
/* 1. */
 for (i=0; i<Elements->Count; i++)
  last_energy += Elements->Items[i]->energy();
/* 2. */
 for(i=0; i<get_dof_number(); i++) {
  dof = (*f_dofs)[i];
  if(!dof->is_constrained()) {
   dof->save_value();
   dof->set_value(dof->get_value() + dof->get_delta());
   energy = 0.0;
   for (j=0; j<Elements->Count; j++)
    energy += Elements->Items[j]->energy();
   f_right_side_force[i]=(energy-last_energy)/dof->get_delta();
   dof->set_value(dof-> get_saved_value());
   dof->set_value(dof->get_value() - dof->get_delta());
   energy = 0.0;
   for (j=0; j<Elements->Count; j++)
    energy += Elements->Items[j]->energy();
   f_left_side_force[i]=(last_energy-energy)/dof->get_delta();
   dof->set_value(dof-> get_saved_value());
  }
 }
/* 3. */
 for(i=0; i< get_dof_number(); i++){
  dof = (*f_dofs)[i];
  if(!dof->is_constrained()){
   F=(f_right_side_force[i]+f_left_side_force[i])/2.0;
   k=(f_right_side_force[i]-f_left_side_force[i])/dof->get_delta();
   ResidualForce=dof->get_load()-F;
   if (k!=0.0)
    dof->set_value(dof-> get_saved_value ()+(ResidualForce/k));
  }
 }
}
```

W procedurze tej można wyodrębnić trzy główne etapy obliczeń. W pierwszym etapie sumuje się w pętli energie odkształcenia elementów rozpatrywanego węzła. Ponieważ funkcja energii odkształcenia jest wirtualna, więc energia odkształcenia zostanie policzona dla dowolnego rodzaju elementu, który jest powiązany z węzłem.

W drugim etapie, dla każdego dostępnego stopnia swobody węzła, funkcja sprawdza czy stopień swobody został utwierdzony. Jeżeli wybrany stopień swobody nie został utwierdzony, to po przesunięciu węzła wzdłuż dodatniego oraz ujemnego kierunku stopnia swobody, a następnie znalezieniu wartości energii odkształcenia elementów otaczających węzeł, obliczane są prawo i lewo stronne siły w węźle. W przeciwnym wypadku obliczenia są pomijane.

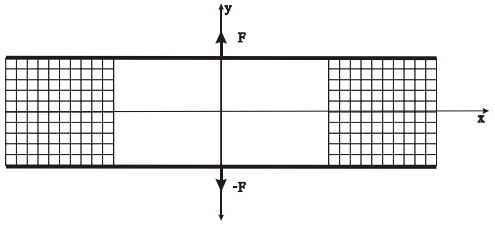
W trzecim i ostatnim etapie funkcja ponownie sprawdza, czy kolejne stopnie swobody zostały utwierdzone, jednak w przypadku gdy stopień swobody nie został utwierdzony, obliczane są siła węzłowa oraz sztywność lokalna, dzięki którym można obliczyć nową pozycje węzła.

Zaletą zastosowanej procedury jest to, że działa ona niezależnie od liczby stopni swobody węzła oraz od rodzaju i liczby elementów go otaczających. Dzięki temu możliwe jest naliczanie energii odkształcenia dla różnych rodzajów elementów stowarzyszonych z węzłem bez zbędnych komplikacji i dodatkowego kodu w programie.

W celu wykonania obliczeń wywołuje się wielokrotnie funkcję *calculate()* dla każdego węzła, którego nową pozycję chcemy obliczyć.

5. MODEL OBLICZENIOWY

Jako obliczeniowy model ciała stałego wybrano obiekt o wymiarach 20 x 20 x 8 mm, w którego środku wydrążono otwór o przekroju kwadratowym. Do górnej oraz dolnej powierzchni modelu przymocowano sztywne nieodkształcalne płyty. Następnie model odkształcono poprzez przemieszczenie płyt względem siebie. Przekrój obiektu reprezentowano przez siatkę, której węzły określały punkty przekroju rzeczywistego (rys. 5).



Rys. 5. Przekrój kostki z widocznym podziałem na elementy skończone

Model wykonano z 3200 elementów tzn. podzielono go na 20 elementów w kierunku osi x_1, 8 w kierunku osi x_2 oraz 20 w kierunku osi x_3. Materiał, który usunięto, posiadał wymiar 8 elementów w każdym z kierunków osi układu współrzędnych. Wymiary Δx_1, Δx_2 oraz Δx_3

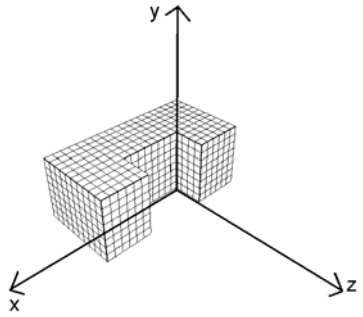
pojedynczego elementu skończonego wynosiły 1,0mm. Każdy węzeł siatki poddano 100 iteracjom.

Jako funkcję energii właściwej odkształcenia przyjęto

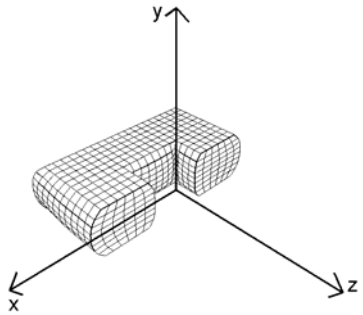
$$U = C_1(I_1 - I_1^V) + C_2(I_2 - I_2^V) + \frac{1}{2}K\frac{(J-1)^2}{J}. \quad (41)$$

Za stałe C_1, C_2, K przyjęto wartości C_1 = 1,92 MPa, C_2 = 0,08 MPa oraz K = 200 MPa.

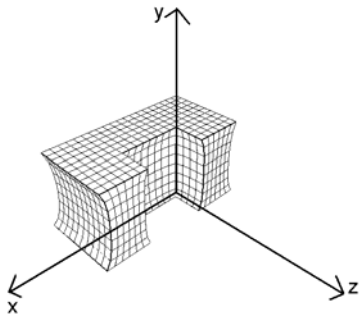
Na rysunkach (rys. 6–8) przedstawiono przekrój modelu obliczeniowego w wybranych fazach odkształcenia. Pokazano deformację obiektu pod wpływem hipotetycznie narastającej siły zewnętrznej F powodującej równomierne przemieszczenie węzłów górnej i dolnej powierzchni zewnętrznej kostki.



Rys. 6. Przekrój modelu w stanie nieodkształconym



Rys. 7. Przekrój modelu ściśniętego o 2,0 mm



Rys. 8. Przekrój modelu rozciągniętego o 2,0 mm

6. PODSUMOWANIE

Z wykorzystaniem uznanych rozwiązań, zaproponowano zastosowanie programowania obiektowego w języku C++ do rozszerzenia systemów MES o elementy skończone modelowane za pomocą funkcji energii właściwej odkształcenia.

Stosując trójliniowe funkcje kształtu oraz funkcję energii właściwej odkształcenia zaprezentowano matematyczny model trójwymiarowego elementu energetycznego typu „brick". Następnie wykorzystując takie pojęcia jak: stopień swobody, węzeł, element i materiał przedstawiono sposób integracji opisanego modelu z systemem komputerowym. Dokonano tego poprzez rozszerzenie standardowych klas o nowe zmienne i funkcje. Przybliżono klasę reprezentującą materiał hiperelastyczny opisywany za pomocą funkcji energii właściwej odkształcenia.

W klasie elementu energetycznego zamieszczono uniwersalną funkcję obliczającą jego energię właściwą odkształcenia oraz funkcję obliczającą jego energię odkształcenia. Następnie wydziedziczono klasę reprezentującą opisany w pracy element energetyczny, w której przedefiniowano funkcję, dzięki której obliczano energię odkształcenia tego elementu.

Przybliżono rozszerzoną klasę węzła, w której zaproponowano metodę obliczającą jego nową pozycję, niezależnie od ilości i rodzaju elementów otaczających węzeł, oraz liczby jego stopni swobody.

Z wykorzystaniem wybranej funkcji energii właściwej odkształcenia przedstawiono prosty numeryczny model ciała stałego wykonanego z materiału hiperelastycznego poddanego dużym odkształceniom.

Zaletą rozwiązań, które zaproponowano, są: łatwość zastosowania w celu rozszerzenia istniejących systemów MES o elementy skończone modelowane energetycznie, zgodność z uznanymi światowymi rozwiązaniami, a także czytelność, zwięzłość, uniwersalność oraz szybkość wykonywania kodów.

LITERATURA

1. **Archer G. C., Fenves G., Thewalt C.** (1999), A new object–oriented finite element analysis program architecture, *Computers and Structures*, Vol. 70, No 1, 63-75.
2. **Athanasiadis A. N., Deconinck H.** (2003), Object–oriented three–dimensional hybrid grid generation, *International Journal of Numerical Methods in Engineering*, Vol. 58, No 2, 301-318.
3. **Attard M. M., Hunt G. W.** (2004), Hyperelastic constitutive modeling under finite strain, *International Journal of Solids and Structures*, Vol. 41, No 18-19, 5327-5350.
4. **Balopoulos V., Abel J. F.** (2002), Use of shallow class hierarchies to facilitate object–oriented nonlinear structural simulations. Finite Elements in Analysis and Design, Vol. 38, No 11, 1047-1074.
5. **Bastian M., Li B.Q.** (2003), An efficient automatic mesh generator for quadrilateral elements implemented using C++, *Finite Elements in Analysis and Design*, Vol. 39, No 9, 905-930.
6. **Besson J., Foerch R.** (1997), Large scale object–oriented finite element code design, *Computer Methods in Applied Mechanics and Engineering*, Vol. 142, No 1-2, 165-187.

7. **Commend S., Zimmerman T.** (2001), Object–oriented nonlinear finite element programming: a primer, *Advances in Engineering Software*, Vol. 32, No 8, 611-628.
8. **Devloo P. R. B.** (1997), PZ: An object–oriented environment for scientific programming, *Computer Methods in Applied Mechanics and Engineering*, Vol. 150, No 1-4, 133-153.
9. **Dolenc M.** (2004), Developing extensible component–oriented finite element software, *Advances in Engineering Software*, Vol. 35, No 10-11, 703-714.
10. **Dubois–Pèlerin Y., Pegon P.** (1998), Linear constraints in object–oriented finite element programming, *Computer Methods in Applied Mechanics and Engineering*, Vol. 154, No 1-2, 31-39.
11. **Dubois–Pèlerin Y., Pegon P.** (1998), Object–oriented programming in nonlinear finite element analysis, *Computers and Structures*, Vol. 67, No 4, 225-241.
12. **Dubois–Pèlerin Y., Bomme P. and Zimmermann Th.** (1991), Object–oriented finite element programming concepts, *Proceedings of European conference on new advances in computational structural mechanics, ed. P. Ladevèze and O.C. Zienkiewicz*, Elsevier Science Publishers, 95-101.
13. **Dubois–Pèlerin Y., Zimmermann Th. and Bomme P.** (1992), Object–oriented finite element programming: II. A prototype program in Smalltalk, *Computer Methods in Applied Mechanics and Engineering*, Vol. 98, No 3, 361-397.
14. **Dubois–Pèlerin Y., Th. Zimmermann** (1993), Object–oriented finite element programming: III. An efficient implementation in C++, *Computer Methods in Applied Mechanics and Engineering*, Vol. 108, No 1-2, 165-183.
15. **Jeremić B., Runesson K. Sture S.** (1999), Object–oriented approach to hyperelasticity, *Engineering with Computers*, Vol. 15, No 1, 2-11.
16. **Ju J., Hosain M.U.** (1996), Finite element graphics objects in C++, *Journal of Computing in Civil Engineering*, Vol. 10, No 3, 258-260.
17. **Karamete B.K.** (1997), Unstructured grid generation and a simple triangulation algorithm for arbitrary 2–D geometries using object–oriented programming, *International Journal of Numerical Methods in Engineering*, Vol. 40, No 2, 251-268.
18. **Kong. X. A. Chen D. P.** (1995), An object–oriented design of FEM programs. *Computers and Structures*, Vol. 57, No 1, 157-166.
19. **Lages E. N., Paulino G. H., Menezes I. F. M., Silva R. R.** (1999), Nonlinear finite element analysis using object–oriented philosophy – Application to beam elements and to the Cosserat continuum, *Engineering with Computers*, Vol. 15, No 1, 73-89.
20. **Lichao Yo, Kumar A. V.** (2001), An object–oriented programming modular framework for implementing the finite element method, *Computers and Structures*, Vol. 79, No 9, 919-928.
21. **Mackie R. I.** (1990), Object–oriented finite element programming – the importance of data modelling, *Advances in Engineering Software*, Vol. 30, No 9-11, 775-782.
22. **Mackie R. I.** (1997), Using objects to handle complexity in finite element software, *Engineering with Computers*, Vol. 13, No 2, 99-111.
23. **Mackie R. I.** (1998), An object–oriented approach to fully interactive finite element software, *Advances in Engineering Software*, Vol. 29, No 2, 139-149.
24. **Mackie R. I.** (2000), An object–oriented approach to calculation control in finite element programs, *Computers and Structures*, Vol. 77, No 5, 461-474.
25. **Mackie R. I.** (2001), *Object–oriented methods and finite element analysis*, Saxe–Coburg Publications, Stirling, ISBN 1-874-672-08-3.
26. **Mackie R. I.** (2002), Using objects to handle calculation control in finite element modelling. *Computers and Structures*, Vol. 80, No 27-30, 2001-2009.
27. **Mackie R. I.** (2004), Extensibility of finie element class system – a case study, *Computers and Structures*, Vol. 82, No 23-26, 2241-2249.
28. **Menétrey Ph., Zimmermann Th.** (1993), Object–oriented nonlinear finite element analysis: application to J2 plasticity, *Computers and Structures*, Vol. 49, No 5, 767-77.
29. **Money M.** (1940), A theory of large elastic deformation. *Journal of Applied Physics*, Vol. 11, 582-592.
30. **Patzák B., Bittnar Z.** (2001), Design of object–oriented finite element code, *Advances in Engineering Software*, Vol. 32, No 10-11,759-767.
31. **Phongthanapanich S., Dechaumphai P.** (2006), Easy FEM – An object–oriented graphics interface finite element/finite volume software, *Advances in Engineering* Software, Vol. 37, No 12, 797-804.
32. **Rivlin R. S.** (1956), Rheology theory and applications. Red. F.R. Eirich., Academic Press, New York, Vol. 1, 351-385.
33. **Rucki M. D., Miller G. R.** (1998), An adoptable finite element modelling kernel, *Computers and Structures*, Vol. 69, No 3, 399-409.
34. **Wegner T.** (1997), Energetyczna metoda modelowania i wyznaczania dynamicznych charakterystyk elementów mechanicznych o silnym tłumieniu. Seria Rozprawy, nr 323, Wydawnictwo Politechniki Poznańskiej, Poznań.
35. **Zimmerman Th., Bomme P., Eyheramendy D., Vernir L., Commend S.** (1998), Aspects of an object–oriented finite element environment, *Computers and Structures*, Vol. 68, No 1-3, 1-16.
36. **Zimmermann Th., Dubois–Pèlerin Y. and Bomme P.** (1992), Object–oriented finite element programming: I. Governing principles, *Computer Methods in Applied Mechanics and Engineering*, Vol. 98, No 2, 291-303.

EXTENSIBILITY OF OBJECT–ORIENTED FINITE ELEMENT CLASS SYSTEM WITH A CONCEPTION OF FINITE ELEMENT BASED ON A STRAIN ENERGY DENSITY FUNCTION.

Abstract: The main purpose of this article is a presentation of the computational method of finite element based on a strain energy density function and its implementation in an object–oriented environment. The original adaptation of the nonlinear finite element is introduced. The different use of the finite element is basing on the old–style framework of classes. Properties of a material are modeled with the modified strain energy density function. The local relaxing procedure is introduced as a solving method implemented in C++ language. The application of the proposed finite element is exposed on the example of computational object made of nearly incompressible hyperelastic material.